

**Development of Smart Devices
in Node.JS**
An Exploratory approach

Master's Thesis in Computer Science
Østfold University College

Anders Skaalsveen

May 15, 2015
Halden, Norway

Abstract

An increasing number of everyday objects with embedded electronics are being introduced to the consumers to enable new services and features. When developing these smart devices, the selection of standards and protocols affects how they perform and which other devices they are able to communicate with. Separating the concerns of communication and smart device functionality, may be appropriate to let experts in each field work on different parts of the functionality.

In this thesis, an exploratory approach is combined with qualitative evaluation in order to study Internet of Things (IoT) and provide answers for how communication related functionality in smart devices for homes can be separated and encapsulated within a Node.JS module. I present how the module incorporates communication functionality, such as local service discovery, data and notification exchange, in addition to how it allows continuous streams to be shared between smart devices. After being implemented, it is evaluated by developers during a series of workshops, where they develop their own smart device prototypes based on the module.

Acknowledgment

A very special thanks goes to my mentor, Harald Holone, for giving me guidance and invaluable feedback through the course of this project. I would also like to thank Caroline Sofie Olsen and Martin Eskerud for creating such a good atmosphere in the lab over the years. Finally, I would also like to thank my family for their unconditional support and encouragement to pursue an advanced degree.

Halden, Norway.

Anders Skaalsveen

Contents

1	Introduction	1
1.1	Project Motivation	2
1.1.1	Personal Motivation	2
1.1.2	Industrial Motivation	4
1.1.3	Academic Motivation	4
1.2	Research Question	5
1.3	Report Outline	6
2	Background	7
2.1	Internet of Things	7
2.2	Web of Things	7
2.3	Standards	8
2.3.1	Representational State Transfer	8
2.3.2	Universally Unique Identifier	8
2.3.3	Comet	8
2.3.4	Web Hooks	9
2.4	Node.JS as a Platform	9
2.5	Protocols	9
2.5.1	WebSockets	10
2.5.2	Extensible Messaging and Presence Protocol	10
2.5.3	Constrained Application Protocol	10
2.5.4	Multicast DNS	10
2.6	Related Work	11
2.6.1	Overview	11
2.6.2	Automatic Configuration	11
2.6.3	Home Networks	11
2.6.4	Adoption of Technology	11
2.6.5	Context	11
2.7	Summary	12
3	Method	13
3.1	Selection of Methods	13
3.2	Exploration	14
3.3	Scenarios	14
3.4	Prototype	15
3.5	Evaluation	15
3.6	Semi-Structured Interviews	16
3.7	Analysis	16

3.7.1	Open Coding	16
3.7.2	Focused Coding	16
4	Home-Thing	19
4.1	Scenarios	19
4.1.1	Scenario 1 - Local Peer Discovery	19
4.1.2	Scenario 2 - Methods of REST	20
4.1.3	Scenario 3 - Sensor Events	20
4.1.4	Scenario 4 - Streaming	21
4.2	Implementation	21
4.2.1	Module Life-Cycle	21
4.2.2	Service Discovery	22
4.2.3	Disconnect Signal	23
4.2.4	Network Interface	24
4.3	Application Programming Interface	24
4.3.1	Properties	24
4.3.2	Subscriptions	25
4.3.3	Streams	25
4.4	Example Prototypes	26
4.4.1	Chat	26
4.4.2	Distributed Volume Control	26
4.4.3	Video Streaming	27
4.5	Documentation	27
5	Results	29
5.1	Workshops	29
5.1.1	Available Smart Devices	30
5.1.2	Tutorial	31
5.1.3	Observations	31
5.1.4	Interviews	32
5.1.5	First Workshop	32
5.1.6	Second Workshop	33
5.1.7	Third Workshop	34
5.2	Prototype Code	34
5.3	Transcripts	34
5.4	Analysis	35
5.5	Findings	36
5.5.1	Developers' Backgrounds	36
5.5.2	Workshop	38
5.5.3	Prototype Development	38
5.5.4	Documentation	39
5.5.5	Home-Thing Module	41
5.5.6	Issues	43
5.5.7	Reflection	45
5.5.8	Summary	46

6	Discussion	49
6.1	Project Structure	49
6.2	Development Platform	50
6.3	Application Programming Interface	50
6.4	Documentation	51
6.5	RQ1 - Discovery	52
6.5.1	Exploration	53
6.5.2	Feedback	53
6.5.3	Summary	57
6.6	RQ2 - Data Exchange	57
6.6.1	Exploration	58
6.6.2	Feedback	60
6.6.3	Summary	61
6.7	RQ3 - Streams	61
6.7.1	Exploration	61
6.7.2	Feedback	62
6.7.3	Summary	64
7	Conclusion	65
7.1	Contribution	65
7.1.1	Research Question 1 - Service Discovery	65
7.1.2	Research Question 2 - Properties and Notifications	66
7.1.3	Research Question 3 - Streams	66
7.2	Reflection	66
7.2.1	Selection of Methods	66
7.3	Future Work	67
	Bibliography	69
	Glossary	75
	Acronyms	77
A	Documentation	79

Chapter 1

Introduction

When something is such a creative medium as the web, the limits to it are our imagination.

- Tim Berners-Lee

Today, an increasing number of home appliances and everyday objects are introduced to the consumer market with embedded electronics. The popularity of body-borne computers such as smart watches and heart rate monitors also creates a focus on seamless communication between the smart devices. The IoT, is referred to by researchers as possibly the next big leap forward, where the devices are all linked to each other and the internet. Haller et al. defines IoT as:

A world where physical objects are seamlessly integrated into the information network, and where the physical objects can become active participants in business processes. Services are available to interact with these 'smart objects' over the Internet, query their state and any information associated with them, taking into account security and privacy issues [28].

In this thesis I refer to smart objects as smart devices. Smart devices can be passive objects or interactive tools assisting the user with tasks in the physical world [28, 32]. As these devices with sensors and increasingly more processing power, emerge in the consumer market, important challenges are revealed. There are a variety of protocols currently used by the different devices, which makes intercommunication between heterogeneous devices a difficult challenge. This also leads to situations where only certain similar devices, as for instance those made by the same company, are able to communicate with each other. Smart objects working alone can give us novel services. However, their true power will only be realized when they can work together [32].

In this thesis I explore the IoT in order to simplify development of software for smart devices on the Node.JS platform. The solution I present in this thesis is only one of many possible solutions. Nonetheless, it demonstrates how it is possible to abstract functionality for communication between smart devices to allow for separation of concerns.

1.1 Project Motivation

In the future, smart devices will in greater extent facilitate creation of automatic behavior by the home owners. They will be able to tailor relations between smart devices of different brands in order to create event driven home automation that extends beyond the home. With new opportunities, a number of sectors will benefit from the IoT. The health sector for instance, will be able to let elderly people live longer in their own homes by using IoT technology. Smart devices with privacy preserving mechanisms can monitor and assist the elderly when it is needed. The smart devices can for instance remind them to take their medicines if they forget. Assistance can also be summoned if the smart devices are equipped with sensors to detect if the elderly falls. The smart devices can be programmed to detect abnormalities to achieve this.

In the following sections, I describe my personal motivation for taking on this project followed by industrial motivation and academic motivation. The research questions are finally presented in Section 1.2.

1.1.1 Personal Motivation

From my perspective, as a developer, I find a number of different use cases for IoT technology. However, I can also imagine it has potential for causing serious problems if world wide adaptation takes place without sufficient security. Miorandi et al. [39] emphasize that security and privacy-preserving mechanisms should be a key feature in IoT devices. Therefore, perhaps security and privacy related features in smart devices should be handled by specialized developers within that area. A separation of concerns may be appropriate, in order to allow communication experts to create some parts of the functionality and other developers to build the rest.

Dynamic Home Environments

With a low-level programming language, the possibilities for what can be achieved with a few lines of code is more limited than in higher-level languages. The complexity of the software increase, and the developer are able to do more general operations with less lines of code. Similarly, in the early phase of IoT, smart devices were limited to simple tasks and were not given the chance to show their full potential. As their complexity and abilities have expanded, and the standardization activities proceed to create common protocols tailored to their use, smart devices will offer simple and clever solutions for achieving advanced home automation functionality.

With homes having IoT technology embedded into surfaces and everyday objects, it will be possible to tailor more advanced automatic behavior based on the inhabitants needs. Homeowners will be able to make advanced behavior such as continuously regulating and enhancing air quality in their indoor environments to help increase productivity and prevent sleepiness. In addition to achieve good air quality, the automatic behavior could also be based on profiles for minimizing power consumption. Smart devices placed in various locations could detect air composition around them and share the data on the internet. Additionally, they could be linked to devices controlling air vents or windows. The data generated by both the air composition devices and the air vent con-

trollers, could be aggregated and analyzed with data from thousands of other homes in order to analyze and study the effects of ventilation. Advanced optimized models for ventilation could then be created and applied to all the homes through software updates.

Smart picture frames, speaker systems and colored lights can for instance be used to create dynamic environments, configured to reflect events around the world. For some people, it is important to only be interrupted when it is important. Events on social media can be interpreted and visualized as colored decorative light in their living rooms in order to discretely indicate new happenings. This way, they can discover the signs of new notifications later when they are ready to seek the information themselves. Colored light and vibration in the floor are both examples of how ubiquitous smart devices can be used for encoding information into their environments.

Controlling the Home

In 2014, I worked on a project together with Caroline Sofie Olsen and Martin Eskerud, where we created a mobile user interface for home automation. The concept of a graph based interface was conceived through user participatory design, involving users already in the earliest phases of the design process. The concept was later developed as an application for Android as illustrated

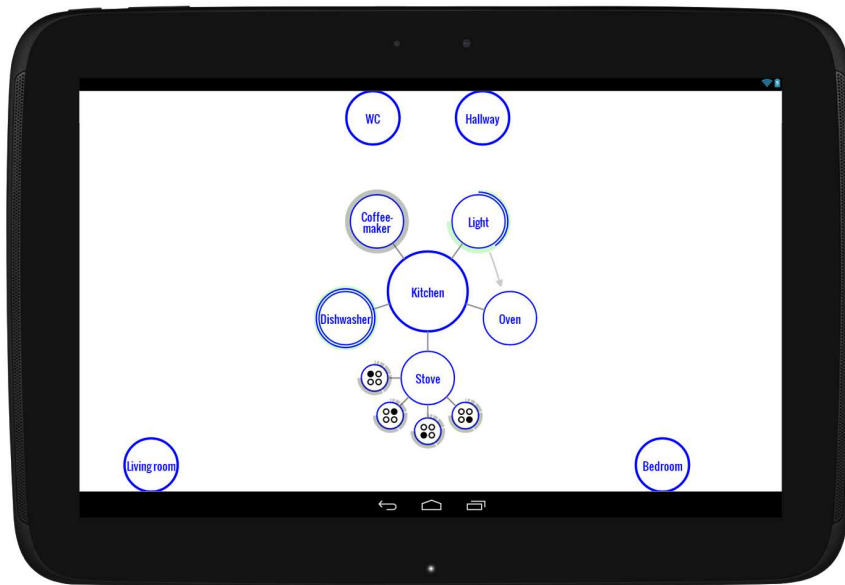


Figure 1.1: Illustration of the *Controlling the Home* application.

in Figure 1.1. In order to display smart devices in the application, they are represented as device nodes, which are connected by edges to their respective rooms nodes. The smart devices are turned on and off when the user taps the device representation. Some of them, as for instance nodes representing sound volume on a smart device, can also be adjusted with a drag gesture that allows for fine-tuning of its value.

As the project was focused on the interface itself, it does not have a working back-end. In this thesis, I therefore wanted to create something that would allow me to connect a back-end to the application and enable controlling of smart devices.

1.1.2 Industrial Motivation

IoT will allow new business sectors to emerge and new products to be created. Embedded electronics and everyday objects will come together to create new products. When the smart devices are created to follow universal standards, it may perhaps also give rise to services similar to app-stores. With a number of interconnected smart devices available, the users will be able to buy, download and install software to get new functionality.

For manufacturers and companies, IoT will bring wide applicability in existing sectors and provide new opportunities. For instance, by adding Radio-Frequency Identification (RFID) to products, or carts moving products around, companies are able to track their journey through the supply-chain and monitor parameters like temperatures and bacterial composition. This allows retailers to keep track of their inventory in real-time and guarantee the required quality final products [39], which will benefit both companies and consumers.

Smart clothes monitoring health parameters, could serve workers in the health sector and the patients by for instance combining various historical data to provide better understanding of each individual's health. With a common underlying communication platform, IoT technologies will open up new business opportunities that will give rise to new cross-cutting applications and services [39].

1.1.3 Academic Motivation

When introducing new technology to the home, an important part of the design is to make it simple to use, as people often are unable to describe problems in their networks accurately [42]. Edwards et al. [18] reported that the adoption of home networks has increased over the last decades and created collisions between technical design and fundamental aspects of human nature. This causes problems for millions of users, threatening their privacy and creating barriers for adoption of next-generation applications in the home.

Miorandi et al. [39] reported in a survey paper from 2012 a clear lack of standardization activities on models, naming and format for IoT in terms of service-layer interfaces and protocols. Today, however, various standardization activities are working on creating IoT specifications. Constrained RESTful Environments (CoRE) working group of the Internet Engineering Task Force (IETF) is one example. They have taken on the task of making a framework for applications using simple resources on constrained networks¹. The working group made the specifications for Constrained Application Protocol (CoAP) which will probably be an important part of smart devices in the future. Although there is a desire for discovery and configuration features in CoAP, CoRE reports they have not yet closed in on a specific approach for this.

Figuring out all the details for the standards might take time. Although the standardization process will happen, it may take years before all decide on

¹<https://datatracker.ietf.org/doc/charter-ietf-core/>

jumping the same bandwagon. Research also proceed in order to create novel solutions for the IoT. Wilde et al. [59] emphasize that it is important not to try solving too many problems at the same time. They argue that the Internet and the Web thrive because they are ecosystems where only the useful and popular foundations grow and develop while the other foundations that remain unused disappear.

1.2 Research Question

Embracing IoT may help improve our lives through novel solutions that we have yet to imagine. New business sectors will emerge as embedded electronics and everyday objects come together to form smart devices. Common standards will be important in order to enable smart devices to interconnect and exchange data. This thesis contributes with a possible solution for separating transfer protocols and low-level communication features from other smart device functionality. The first research question is therefore:

Research Question 1

How can smart devices discover and connect with similar peers on a local network without depending on a common server?

Given that the smart devices are able to connect within a home network, the next challenge is how smart devices can be enabled to manipulate each others resources and send event notifications to similar devices. To address the problems of heterogeneous devices, I will focus on using Web technology. In the second research question I ask:

Research Question 2

How can smart devices in home networks exchange data, send notifications and affect each other by utilizing proven Web technology for transferring states?

By solving the second research question, I enable smart devices to exchange simple data resources such as sensor values and power states. However, as it might be desirable to also transfer other resource types such as audio and continuously updated real-time sensor data, the need for transferring resources as for instance data streams, brings us to the third and final research question:

Research Question 3

How can data streams be utilized by smart devices in order to transfer continuously updated data from sensors?

The first phase of the project is focused around exploration and prototype development. In order to answer the above mentioned research questions, I first create four scenarios. The scenarios are based on how I can give a possible answer to each research question. They also reveal prototype requirements, as for instance what kind of information that is needed in order to exchange data between smart devices. When the prototype is created, the project reach its second phase, which is focused on qualitative evaluation. Developers are invited to workshops where they develop their own software for smart devices by using the module as a building block. After having taken part in development, each developer is interviewed to share their thoughts about the prototype.

1.3 Report Outline

The rest of this thesis is organized as follows: In Chapter 2, I describe key concepts, standards, the prototype platform and protocols, before related work is reviewed. Furthermore, the selected method are described in Chapter 3. Chapter 4 encloses details regarding the prototype and is followed by Chapter 5, which describe the results obtained from the workshops. In Chapter 6, the results and findings from the exploratory phase are discussed in light of the project's research questions, before I conclude in Chapter 7.

Chapter 2

Background

With great power comes great responsibility.

- Uncle Ben

The Internet has experienced enormous growth over the last decades. With increasingly more people connected, new ideas of how it can be evolve further arise. In the following sections, I present a selection of relevant concepts, standards and the prototypes platform in addition to important protocols and related work. In the end of this chapter a short summary can be found.

2.1 Internet of Things

The basic idea behind Internet of Things (IoT) is that physical objects are connected to the internet and each other [37]. The smart devices can for instance be sensors, actuators, mobile phones and typical everyday objects that surrounds us [2, 46]. The IoT emerges, as an increasing number of everyday objects are embedded with electronics and are able to interpret physical phenomena in their surroundings [39]. With globally connected smart devices, we will be able to interact with the Internet in new ways. Visions of the IoT typically focus on devices with limited resources in terms of battery and processing power [2]. Various standardization activities exists in order to overcome the current challenges of IoT. Constrained Application Protocol (CoAP) for instance, is a protocol created by the Internet Engineering Task Force (IETF) Constrained RESTful Environments (CoRE) working group, which is a specialized Web protocol for devices with constrained resources [12, 50]. Another example is the IPv6 over Low power Wireless Personal Area Networks (6LoWPAN) mesh network standard, that also was defined by the IETF as a result of the uprising of the IoT and the introduction of IP based Wireless Sensor Networks (WSNs) [47]. Zhao and Sudhaakar think WSNs will play an important role at the edge of IoT [60].

2.2 Web of Things

In order to avoid compatibility issues, open standardized interfaces and communication methods are important. So far, research on the IoT have mainly

focused on establishing connectivity in constrained network environments [26]. In Web of Things (WoT) however, which is a concept that derived from IoT, the focus is on using Web protocols. This simplify integration of heterogeneous networks and facilitate rapid development of web services [8, 9]. Lightweight Web servers exposing Representational State Transfer (REST)ful interfaces are embedded into smart devices in order to view and manipulate shared resources. This is done with the four main methods of Hypertext Transfer Protocol (HTTP) [26], detailed in Table 2.1. With increasingly more powerful hardware available, it becomes in greater extent possible to create all kinds of smart devices with full networking capabilities.

2.3 Standards

In this section, a collection of standards that are important to this project will be described. While I in this section focus on the technical details of the standards, they are further discussed in Chapter 6 in light of the implementation of the prototype in Chapter 4.

2.3.1 Representational State Transfer

REST is a stateless architectural style that sets constraints on how services are built for the Web. It was defined by Fielding to support high performance and scalability [62]. By separating the user interface from the data storage, it improves the portability of the user interface across multiple platforms and allows components to evolve separately [22]. In REST, the *CRUD* design pattern is translated to the main methods of HTTP [43] as shown in Table 2.1.

	Create	Read	Update	Delete
HTTP Method	POST	GET	PUT	DELETE

Table 2.1: Translation from the CRUD design pattern to REST methods.

REST is concerned with how Web components interact with each other in order to view an manipulate resources. It does not set restraints on how the syntax of the resources should be [22]. However, common formats used in RESTful services are for instance Extensible Markup Language (XML) and JavaScript Object Notation (JSON).

2.3.2 Universally Unique Identifier

Universally unique identifiers (UUIDs) are also known as Globally unique identifiers (GUIDs). They are 128-bits strings that are designed to be uniquely generated without a central authority [35].

2.3.3 Comet

Comet is an umbrella term for pushing content from a server to a client utilizing HTTP streaming or long polling [46, 36]. Long polling is done by trying to hold the HTTP connection open, creating a pending request that waits for a

response. The response is postponed until there are new content to be delivered, which will also typically trigger the next long poll request. The drawback of this approach is that it creates header overhead for each request and for each response. Additionally, it is unable to push content in the time between a response is sent and the next request is received [36].

HTTP streaming, on the other hand, keeps the request open indefinitely, even after pushing content to the client. This method relies on the capability of the server to send several chunks of data in the same response without terminating the connection. Both HTTP/1.0 and HTTP/1.1 can be used for this purpose. Additionally, HTTP allows for intermediaries, such as proxies and gateways, but does not specify requirements for forwarding partial responses. The intermediaries may also repackage the data chunks, which may cause problems for the client receiving them. Furthermore, in order to handle the growth of memory usage in the client software, the connection must occasionally be restarted [36].

2.3.4 Web Hooks

Web Hook is a design pattern for creating a simple subscription feature enabling users to define callbacks on the Web [14]. By using Web Hook, servers can push event notifications to other servers with HTTP *POST* requests [30]. The incoming request can be connected to a callback in order to trigger remote events.

2.4 Node.JS as a Platform

Node.JS is an open source framework built on top of Google's v8 JavaScript engine for Google Chrome [56, 44]. JavaScript is a light-weight programming language often used in Web browsers [23]. The v8 engine uses an event driven and non-blocking IO model to handle concurrency. Its server applications can be created and executed from command line or Unix Shell [56, 44].

The package manager NPM allows for easy installation and publishing of third-party modules from a large and growing repository [44, 56]. Each module has a manifest file that describes its name, version details and dependencies [56]. In addition to the third-party modules, Node.JS has built-in core modules with support for file system operations, HTTP services, and assertion testing among other things [56].

In node applications, the environment variables in shell are available through the object *process.env* [44]. It is also possible to listen to signals shutting down the application, which can be used to close down the application gracefully [44]. The use of this functionality is later described in Section 4.2.3.

2.5 Protocols

In the following sections, relevant protocols to this project will be described. Most of the detailed protocols were ruled out as components for the prototype, during the project's exploratory phase. However, the details are still relevant to the discussion in Chapter 6.

2.5.1 WebSockets

The WebSocket protocol is independent from HTTP, except for using HTTP in the initial handshake, and is layered on top of Transmission Control Protocol (TCP) [20]. It was designed to replace the HTTP workarounds that were created to push data from servers to clients, as described in Section 2.3.3, by exposing web scripts to socket connections. WebSockets can share port with HTTP servers and use a HTTP Upgrade request as handshake [20]. The handshake is mandatory and require HTTP/1.1 or higher. HTTP Upgrade requests allows clients to specify which protocols they want to use in order switch from using HTTP [21]. The WebSocket protocol creates a bidirectional communication between a client and a server. The data is transferred in frames that supports both text and binary content. Special frames are used for keeping the connection alive and for closing it, which can also contain data [20]. HTTP/2.0 and the WebSocket protocol among others will to be better suited than HTTP/1.1 for streaming continuously updated data as they support pushing content from the server side [59].

2.5.2 Extensible Messaging and Presence Protocol

Extensible Messaging and Presence Protocol (XMPP) is an open standard that use TCP to send real-time XML messages and are used in for instance instant messaging applications such as Google Talk [48, 26]. It was originally developed by the Jabber open source community and was revised and formalized by IETF under the name XMPP in 2004 [34, 49]. The XML messaged are referred to as *XML stanzas* and are transferred in XML streams [48].

2.5.3 Constrained Application Protocol

CoAP is a web transfer protocol in the application layer, which utilizes User Datagram Protocol (UDP) [33]. It was based on the HTTP protocol and was designed to be more resource friendly in order to address the special requirements of Machine to Machine (M2M) applications in constrained environments [12, 51, 12]. CoAP was designed to bring REST to IoT, and is expected to become the standard communication protocol for constrained smart devices in the future [11]. Furthermore, it features multicast support, asynchronous message exchange and resource discovery [52]. Resource discovery in CoAP use the path `/.well-known/core` in order to share information. This follows the RFC 5785 specification for metadata locations, which was created to avoid collisions between *well known* addresses used by discovery mechanisms and Web crawlers for instance [40].

2.5.4 Multicast DNS

In local networks without a Domain Name System (DNS) server, the Multicast DNS (mDNS) protocol can be used with standard DNS queries, to receive a list of desired instances [55, 15]. mDNS sends both requests and responses using multicast. By sending the packets to all devices subscribing to the multicast address, hosts on the network are able to keep their caches up to date [15, 10]. The record names used in mDNS require little or no administration and

configuration to be set up [10]. The mDNS protocol can be used both in IPv4 and IPv6 networks [27].

2.6 Related Work

During the course of this project, I found a number relevant and interesting studies on home networking, IoT and WoT, in addition to other areas in the edge of the project scope. The focus in this thesis, however, had to be refined to a concrete set of research questions. In the following sections, I describe the most relevant work within the project scope that I found to be a source of inspiration.

2.6.1 Overview

Miorandi et al. [39] presented a survey article giving an overview of IoT and its research challenges. They see IoT as a major trend that may represent the next big leap forward in the Information and Communication Technologies sector. They refer to smart devices as physical objects associated to at least one name or address with communication abilities. In addition it should possess a unique identifier and may also be able to sense physical phenomena.

2.6.2 Automatic Configuration

In The Internet of Things Journal, Cirani et al. [11] provide an architecture for service discovery with focus on self-configuration and scalability. They present ways to discover smart devices both locally and globally, using a gateway device to protect the network from the outside and to keep track of local smart devices and their available resources.

2.6.3 Home Networks

Poole et al. [42] studied how to design better network environments in the home. They found that appropriate tools for home network management is important to help householders understand and manage their digital devices. These tools should take into consideration issues of the system's appliances change history in addition to the spacial representation and householder routine.

2.6.4 Adoption of Technology

Venkatesh and Brown [57] studied adoption of technology in American households. They found that the adopters invested in computers to increase the effectiveness of household activities, to get pleasure or for the social status achieved by adopting an innovation. The non-adopters were influenced by fear of rapid change in technology making their computer obsolete.

2.6.5 Context

A central issue for Human-computer interaction (HCI)-research has been how to make devices behave in the correct way based on the current setting they are in, and adapt to the new settings when they are moved [17]. This is

especially important when computation is moved off the desktop to ubiquitous smart devices. Dourish [17] reports in his paper that context should not be considered as a representational problem. Instead it is a problem having to do with interaction. Context should be seen as a relational property between objects and activities that is particular to each occasion. The contextual features are defined dynamically in the course of the activity, and is continually being renegotiated and defined.

2.7 Summary

The IoT is about embedding electronics into everyday objects to allow local and global communication. WoT builds on principles of IoT, with focus on Web protocols and REST, to overcome the communication challenges. REST is stateless and separates the user interface from the data storage. The four main methods of HTTP is used in order to allow clients to manipulate representational objects often formatted in XML or JSON. To enable smart devices to distinguish between each other UUIDs can be created without requiring a central authority.

Before WebSocket, a number of workarounds for pushing content from servers to clients over HTTP were created. These are known under the umbrella expression Comet. Web Hook allows servers to subscribe to events on other servers by specifying URLs for where notifications in the form of HTTP requests, can be sent to when certain events occur.

The Node.JS platform allows server side applications to be written in the programming language JavaScript. It comes with a package manager that allows open source modules to be shared with the community.

WebSocket is layered over TCP and can run in parallel with a HTTP server using the same port. It creates a bidirectional communication between server and client and are able to transfer both binary data and text. CoAP is another transfer protocol which is based on HTTP. It is expected to become the standard for communication within constrained networks. mDNS can be used instead of DNS to do service discovery without requiring a central server. It sends multicast packets in order to communicate with other devices in the network.

In Section 2.6, I described related work with focus on the state of IoT, automatic configuration and home networks. Additionally I also highlighted studies of technology adoption and definition of context in HCI research.

In this chapter, I described important terms and standards in addition to the platform Node.JS, considered protocols and related work. In Chapter 3, the focus will be on the methodology of the project.

Chapter 3

Method

The good thing about science is that it's true
whether or not you believe in it.

- Neil deGrasse Tyson

In this project the paradigm of the Internet of Things (IoT), and its smart devices were investigated through exploration, prototype development and evaluation. The three research questions, introduced in Section 1.2 are listed as follows:

RQ 1: How can smart devices discover and connect with similar peers on a local network without depending on a common server?

RQ 2: How can smart devices in home networks exchange data, send notifications and affect each other by utilizing proven Web technology for transferring states?

RQ 3: How can data streams be utilized by smart devices in order to transfer continuously updated data from sensors?

In this chapter, the methods for the development and evaluation phases will be explained with a focus on theory. The research design in this project includes both exploration and evaluation, which is explained in Section 3.1. In the exploration phase, I created scenarios to find the requirements of a prototype that was later developed. The prototype was a module for the Node.JS platform and is also referred to in this thesis as *the Home-Thing module*.

3.1 Selection of Methods

In this project, I combined an exploratory approach with qualitative evaluation in order to learn more about IoT, and purify the project focus to a concrete set of challenges. I found the exploratory method to be appropriate, as I wanted to expand my knowledge about the intercommunication between smart devices. Having years of experience with Web development, I decided to define

and create the Home-Thing module myself. By involving developers in a series of workshops, I was able to test my ideas and get feedback on the structure from other perspectives and get suggestions for future improvements.

3.2 Exploration

Befring [4] describes exploratory research as being typical for feasibility studies and often used by students when writing their master's theses. The exploratory approach has great relevance in the early phase of a study when empirical data is obtained and analyzed. Exploration can for instance be used in small projects, where the research process is tested by gathering and analyzing data, in order to make conclusions. Exploration can however, also be used in a larger investigations, where the researchers explore new territory using an inductive approach. In this project exploration was used as a door opener into the field of IoT. By reading up on, and studying the development and the current challenges within the research field, I was able to build a more comprehensive understanding of the domain.

Exploratory research studies do not start with a problem or hypothesis [6]. The intention is to gain familiarity with the problem area, in order to later develop hypotheses and to identify problems that need to be further investigated [29, 41]. During the exploration phase, I became familiar with various home automation systems, which share common features with the IoT's smart devices. Furthermore, I investigated relevant research with focus on the motivation behind IoT research.

Exploratory research usually utilize qualitative methods and is used in order to form research questions in an initial stage [53, 54]. According to Shields and Rangarajan [54], exploratory research is loosely structured and often associated with a case study. Additionally, it should facilitate a plan of action and have the openness for new evidence characterized by exploration. Further, they describe five micro-conceptual frameworks and ties the research purpose of exploration together with the framework of Working Hypotheses. The working hypotheses is based on literature and experience. The statements of expectations can both be rational and non-rational. Either way, it must be possible to collect evidence that supports or fails to support the expectations.

The hypotheses for what smart devices would require, formed during the exploration phase and helped me refine the research questions. Following, scenarios was created for each research question, in order to acquire specifications for the Home-Thing module.

3.3 Scenarios

The basic idea of a scenario is a story which is told from the point of view of one or several people [1]. It describes an episode of envisioned use that leads to an outcome [45]. Alexander and Neil [1] describes scenarios as a powerful antidote to the complexity of system development. Directly or at least indirectly, a scenario contains actors with goals or objectives, information about the environment and sequences of actions and events [25]. It is the natural ability of people to easily detect inconsistencies, omissions and threats, that makes

scenarios so powerful [1]. Scenarios can be expressed in various forms, as for instance text, storyboards and video mock-ups [25]. When system design employs a detailed scenario, it can be used as a brainstorming tool which can lead to development of alternative solutions [25, 45]. It changes the development focus from defining system operations to describing how the system will be used to accomplish tasks and activities [45]. Scenarios can be used in all phases of system development, for different purposes [1]. For instance, it can be used to identify and plan evaluation tasks [45]. When creating a test scenario, it should not contain ifs and buts. Instead it should be a simple and straight forward sequence of steps that can be repeated, yielding the same results every time [1]. Scenario based design is a family of techniques used in the early phases of development in order to describe a system [45]. In requirements engineering, scenarios are one of the elements used, where the focus is on bringing out users' requirements through analysis [7, 25].

In this project I created a number of scenarios based on the research questions in order to establish the requirements of the Home-Thing module. These are detailed in Section 4.1.

3.4 Prototype

In much of computer science literature it is realized that what works on paper does not necessarily work in the real world [38]. Development of systems can be thought of as proof by demonstration [41]. According to March and Smith [38], Information Technology (IT) researchers deal with artificial phenomena in studies that can be classified into natural science and design science. Natural science is aimed at understanding the reality and includes research in physical, biological, social, and behavioral domains. Design science instead attempts to create things that serve human purposes, and is a key activity in for example architecture, engineering and urban planning. Natural scientists develop concepts and language to characterize phenomena and create theories. Design scientists produce and apply knowledge of tasks and situations in order to create effective artifacts that can be constructs, models, methods, and implementations. My contribution in this project belongs to the design science, as I create a module serving as a foundation for smart device software to build upon.

3.5 Evaluation

The evaluation of a prototype determine whether or not any progress has been made and requires development of metrics and ways to measure the prototype according to those metrics [38]. In order to evaluate the Home-Thing module, I conducted a series of development workshops with students as developers. In the workshops, they took first part in a short tutorial with focus on Node.JS and JavaScript. They were introduced to the Home-Thing module and given links to the Application Programming Interface (API) documentation. The developers were each given a smart device to develop on, having the module already installed. Following, they were presented to a number of tasks and encouraged to explore the API and the documentation to complete these. When the de-

velopment session was completed, each developer took part in semi-structured individual interviews to obtain feedback and discuss their thoughts about the module. The test setting will be further described in the first sections of Chapter 5.

3.6 Semi-Structured Interviews

When conducting semi-structured interviews, the questions are open for interpretation by the informant. Semi-structured interviews requires more planning and preparation of the questions than structured interviews. It does often also require improvisation during the interview in order to give appropriate response to the informants' answers. However, under the right conditions, a semi-structured interview may give more in return than a fully structured interview [58].

3.7 Analysis

In this project, the Home-Thing module was tested in three workshops by developers. After developing their own prototypes, all participants were interviewed. Both the workshops and the interviews were recorded and later transcribed. The transcripts were then analyzed in a process of inductive content analysis. Inductive content analysis includes open coding, creating categories and abstraction [19].

3.7.1 Open Coding

Open coding, also called *initial coding* [24], is a process in which data is interpreted and conceptualized by a researcher [13]. The researcher carefully examines the data word-by-word, line-by-line or phrase-by-phrase and considers all possible meanings in order to conceptualize the data [13, 16, 24]. The concepts arising from the text are descriptive names of for instance events, activities, functions and relationships [16]. Corbin and Strauss defines concepts as:

Words that stand for ideas contained in data. Concepts are interpretations, the product of analysis [13].

These can either be constructed codes by the researcher based on the meaning of the sentences, or "*in-vivo codes*", which are selected words used by the participants [13, 31]. In the process of open coding, the codes can be set temporarily, and modified at a later time in order to improve their fit with the data [24].

3.7.2 Focused Coding

When the open coding process stops yielding new codes, links between the codes are reviled and some codes stands out as more significant than others, more focused coding should be pursued [5]. In Focused Coding, the concepts are refined and grouped into broader and more abstract categories. A categories is described by Corbin and Strauss [13] as being a higher-level concept, grouping lower-level concepts beneath it based on their common properties. Through

repeated iterations, the links between the codes and the categories begin to emerge and the researcher can start looking up the data associated with a given code in order to compare and determine their fit [5].

Chapter 4

Home-Thing

By staying neutral, I end up being somebody that everybody can trust. Even if they don't always agree with my decisions, they know I'm not working against them.

- Linus Torvalds

In this chapter the features of the Home-Thing module and the tools used to create it is described. The module serves as proof by demonstration and the results from its evaluation is described in Chapter 5. Before describing the technical features of the Home-Thing module, I will first describe the scenarios used for determining its specifications. Following, the technical sections, selected smart device prototypes and the documentation will be described. The full documentation is available in Appendix A.

4.1 Scenarios

The following scenarios were created with the project's three research questions in mind. The first scenario regarding local peer discovery reflects Research Question 1. The second and the third scenario concerning data transfers and event notifications reflects Research Question 2. Finally, the fourth scenario that make use of streaming reflects Research Question 3. It should be noted that the following scenarios are based on the module working under ideal conditions.

4.1.1 Scenario 1 - Local Peer Discovery

In the first scenario, two smart devices in a home network are observed, each based on the module. The two devices are both connected to the network router and have either received IP addresses from the router or been manually configured, as this functionality is outside the project's scope. However, the smart devices do not acquire any knowledge about how many other smart devices there are on the network. They also do not know what the IP addresses of the other smart devices are. In order to find out, and share its own information with the other devices on the network, they start service discovery and finds similar smart devices currently on the network. The data from each responding device

contains a link to a data resource that contains more specific information about it. Through service discovery, both sides are able to retrieve each others IP addresses and are able to send requests to acquire more information about the other smart devices. The information available consists of a unique identifier in addition to name and a human readable description of the smart device.

4.1.2 Scenario 2 - Methods of REST

The smart devices are able to find similar devices through service discovery as described in Section 4.1.1. Similar to objects in Object-Oriented Programming (OOP), each smart device has a set of properties that holds data. The properties can both be read and modified by other smart devices. A smart device with a list property that represents a music playlist for instance, can share this playlist with the other smart devices on the network and allow them to modify it. Another smart device operating as a user interface can be used to manipulate the entries. In order to accomplish this, the Home-Thing module utilize the four main methods of Hypertext Transfer Protocol (HTTP), which is GET, PUT, POST and DELETE.

To retrieve the playlist and display it, the user interface sends a GET-request to the device that owns the playlist. The user then utilize the interface to add a new song to the playlist. Now, the interface sends a POST-request in order to create a new entry in the playlist with the name or URI to the song. Later, the user finds out that the added song was misspelled. Therefore, in order to correct it, the correctly spelled string is entered into the interface and sent as a PUT-request to the playlist device, in order to update the list entry. After updating the playlist, the user regrets adding the song and wants to delete it. By utilizing the interface, it finally sends a DELETE-request to the playlist owner, which removes the song.

4.1.3 Scenario 3 - Sensor Events

In this scenario there are two smart devices, also known to each other through service discovery as described in Section 4.1.1. In this scenario, I focus on two smart devices connected to a light and a motion sensor. The first device is able to turn on and off the light, and the other listens to data from a motion sensor. The motion sensor changes its value from 0 to 1 for 3 seconds when it detects motion in the room.

The light will shine only while the motion detector's state is 1. This is achieved through a subscription mechanism that enables the smart device connected to the light to subscribe to changes in the motion sensor's state through the other smart device. In order to create the subscription, first the smart device with the light sends a HTTP POST-request to the other smart device, specifying which attribute it wants to receive updates on, and which path it should be sent to. The path is specified in a URL. The smart device with the motion sensor will add this information to its list of subscribers and iterate it in order to notify all subscribers about state change. An illustration of the subscription mechanism is shown in Figure 4.1.

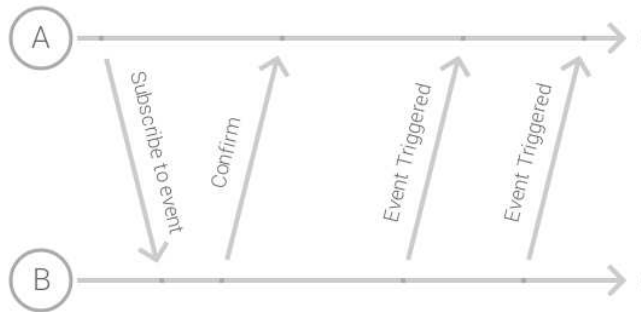


Figure 4.1: Property subscription between two smart devices.

4.1.4 Scenario 4 - Streaming

In this scenario a homeowner creates a door phone with video in order to watch visitors outside the house, pressing the doorbell button. One of the smart devices is mounted on the outside wall next to the entrance. This device is connected to a surveillance camera, a microphone and the doorbell. The other smart device inside the house has a screen, a set of buttons and a microphone in order for the homeowner to see who is outside before letting them in by pushing one of the buttons. The video from the smart device outside is recorded using the program `ffmpeg`¹, which allows it to be piped to the input stream of other processes. The video player on the other smart device inside the house, allows playing video from an input stream. A socket stream is therefore established between the two devices in order to transfer the data stream from the camera to the video player.

4.2 Implementation

Having completed the scenarios in Section 4.1, I set out to create an implementation for enabling the described functionality between smart devices. Node.JS allows applications to be written in JavaScript, which has become increasingly popular [3]. Node.JS is an open source project built on Google's v8 JavaScript engine. It has a dedicated package manager called NPM for publishing and installing modules. The module I created was therefore built on Node.JS and published with the name `Home-Thing`.

Figure 4.2 depicts a simple overview of the module design. In the following sections I will describe each component and the underlying functionality of the `Home-Thing` module.

4.2.1 Module Life-Cycle

The `Home-Thing` module's life-cycle consists of five steps, listed as follows:

1. **Configuration** When the module starts, it stores its configuration in an object. The object holds the name and description, provided by the

¹<https://www.ffmpeg.org/>

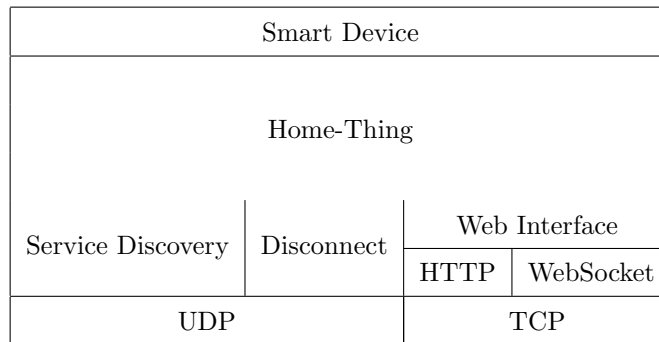


Figure 4.2: Components of the Home-Thing module.

smart devices that are built on top the Home-Thing module. In addition it holds details about subscriptions and the unique identifier, generated by the `node-uuid` module. The Universally unique identifiers (UUIDs) are uniquely generated for each instance of the module, in order to allow smart devices to distinguish between different sessions of the smart devices. More about UUIDs is detailed in Section 2.3.2.

2. **Web Interface** The next step is to start the Web interface, which utilizes both HTTP and WebSockets. Instead of using port 80, which is default, it uses port 4004 for HTTP and 4014 for WebSocket. In theory, both protocols are able to use the same port, as mentioned in Section 2.5.1. This is further discussed in Section 6.7.1.
3. **Listeners** Before sending broadcasts, the module sets up a listener for incoming User Datagram Protocol (UDP) packets. This listener allows it to receive broadcast announcements from other smart devices when they search the network for other similar devices.
4. **Broadcast Announcement** The last step of the initialization phase is to broadcast UDP packets on the network. This allows devices based on the module to discover IP addresses and port numbers of other similar devices.
5. **Shutdown** When the process shuts down, listeners are stopped and a dedicated UDP packet is broadcasted to all devices on the network containing a disconnect message. This message is not sent when the process crashes, as discussed in Section 6.5.

4.2.2 Service Discovery

The first research question concerned how to connect smart devices through distributed service discovery. This is the process in which devices are able to find other devices on the network. The approach used in the Home-Thing module, was to create a custom announcement based implementation as a component in the module.

This implementation utilizes UDP broadcasting on a predefined port, which was chosen to be port 50000. This port is outside the range of well known and

registered ports and is the only port used by the module that will need to be the same in future versions, in order to enable backward compatibility.

Before broadcasting its presence to the other smart devices on the network, the module first generates an UUID using the module `node-uuid`. The UDP packets sent to the network contains a JavaScript Object Notation (JSON) object with one link for each Web interface on the smart device. Information about each device can be retrieved through the URL of the HTTP interface. This information can help smart devices classify other devices in order to determine whether or not the device is relevant to the current context, as described in Section 2.6.5. Each broadcast packet is answered by the receiving smart devices with a UDP unicast packet containing similar information about themselves. This way, each new smart device on the network is able to exchange information about them selves and receive information about other devices, which is possible with only two UDP packets for each device on the network. The information shared through service discovery is also available to developers through the Application Programming Interface (API) described in Section 4.3.

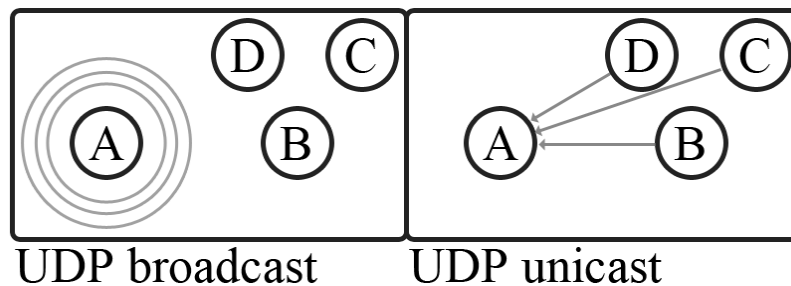


Figure 4.3: Announcement based broadcast for service discovery.

The Home-Thing module sends UDP packets using one of the core modules in Node.JS, *datagram*. From this module, the IP address of the announcing Home-Thing module instances are available to the receiving ones. Therefore, only relative paths are used in the packets exchanges between the devices.

Although the advantage of using this approach is that it needs no configuration from the user, the drawback is that broadcasting is limited to a single hop [61]. Homes with multiple routers and subnets will therefore have divided networks of smart devices. Figure 4.3 illustrate the communication between a device searching for other controllable devices on the network.

4.2.3 Disconnect Signal

UDP packets was not only used for announcement based broadcasting. After having built a number of prototypes based on the Home-Thing module, it became apparent that the smart devices would need to know when other devices were no longer available. By using a listener for exit signals, I was able to create a callback executing code in order to send the disconnect messages to the other smart devices on the network. Sending individual notifications via the module's Representational State Transfer (REST) API would also be possible, but using UDP to broadcast the packet made it more efficient decreased the response time

in Unix Shell when shutting down the smart device instance.

4.2.4 Network Interface

In order to create a Web interface, the *express* module for Node.JS was used. It hosts a HTTP/1.1 server on the specified port and allows setting of callbacks associated with specific paths. This allowed the Web server structure in the module to easily be built on the principles of REST. HTTP requests to other smart devices was sent using the *request* module. This allowed the Home-Thing module to both function as a server and a client.

Streaming functionality was enabled with WebSockets, which allows both text and binary data to be exchanged between a client and a server. More details about streams will be described in Section 4.3.3.

4.3 Application Programming Interface

The module enables communication and data exchange between smart devices with different functionality. The underlying network functionality is hidden beneath an API in order to separate protocol specific communication from the other functionality of the smart devices.

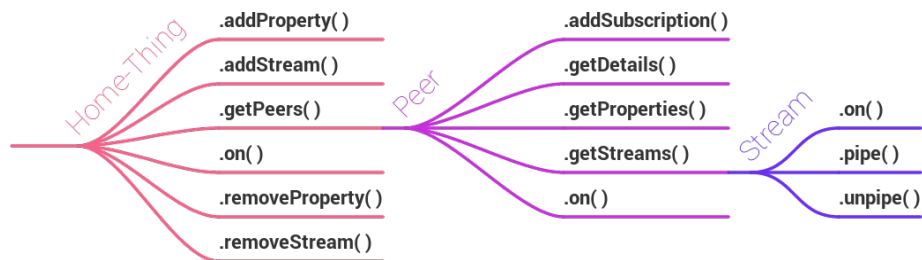


Figure 4.4: Illustration of methods in representational objects.

The developers using the module, utilizes instances of representational objects, which hides unnecessary complex details, such as the unique identifiers and network addresses. The module creates local objects for representing other smart devices, as well as their properties, subscriptions and streams. In the module API, another smart device is referred to as a *peer*. Peers can be received by using the *getPeers()* function, which returns all peers. They can also be received through a callback for the *new peer* event. This callback is connected to the event using the *on()* function in the Home-Thing root object. Figure 4.4 illustrates the methods available in three of the representational objects from the API. It also illustrates an example of how the objects are connected to each other.

4.3.1 Properties

Each smart device using the Home-Thing module as a building block can define properties similar to properties of objects in OOP in order to share data with other smart devices on the network.

In order to define a property, the *addProperty()* function can be used to declare callbacks for set, get, add and remove, where the last two are used for lists and objects. The code in Listing 4.1 gives an example for how to share a simple value, that could for instance affect the volume of the device. When receiving representational objects for external properties from other smart devices, their available methods reflect callback functions that was declared on that device.

```
1 ...
2 var volume = 0;
3
4 var volumeProperty = app.addProperty("volume", "number", "Sound volume", {
5
6     // The get method must use a callback to return the value.
7     'get' : function(callback){
8         callback(volume);
9     },
10
11    // The set method must also use a callback to indicate success.
12    'set' : function(value, callback){
13        volume = value;
14        callback();
15    }
16
17 });
```

Listing 4.1: Example code for adding a property.

4.3.2 Subscriptions

Subscriptions allows smart devices to receive updates when external properties change. To create a subscription for a property, the *addSubscription()* function in the representational peer object can be used. As illustrated in Listing 4.2, this function takes the name of the property and a callback as parameter.

```
1 ...
2 // Create a subscription to the external property named volume
3 peer.addSubscription("volume", function(data){
4
5     // Do something interesting with the data
6
7 });
```

Listing 4.2: Example code for creating a subscription.

4.3.3 Streams

Sometimes it is convenient to be able to transfer data as streams. Video and files are some examples where the module uses the WebSocket protocol to achieve this by transferring continuous data or chunks of text. In order to establish WebSocket connections, the module utilize three NPM modules named *socket.io*², *socket.io-client*³ and *socket.io-stream*⁴. The two first modules are used to establish a WebSocket connection between two devices and the third module allows

²<https://www.npmjs.com/package/socket.io>

³<https://www.npmjs.com/package/socket.io-client>

⁴<https://www.npmjs.com/package/socket.io-stream>

streams to more easily be piped through it. In Listing 4.4, example code shows how to share the video stream from a connected web camera using the Home-Thing module.

4.4 Example Prototypes

During the development of the Home-Thing module, it was important to design the API from a smart device developers perspective, in order to allow it to be sufficiently flexible. Therefore, I created a number of smart device example prototypes, that could also illustrate the capabilities of the module for developers unknown to it. In the following sections, I describe a selection of these prototypes.

4.4.1 Chat

In Node.JS, the standard input and standard output of the Unix Shell are available as streams in the global object *process*. Text entered in Unix Shell can be read from the *process.stdin* stream and written to the Unix Shell through *process.stdout*. By connecting these streams to the Home-Thing module, I was able to create a simple Peer-to-Peer (P2P) chat client with only a few lines of code, which is shown in Listing 4.3.

```
1 // Initializing the smart device
2 var description = "A Home-Thing based chat client.";
3 var app = require("home-thing")("P2P-Chat", description);
4
5 // Publish keyboard input as stream
6 app.addStream("chat", "P2P Chat stream", process.stdin);
7
8 app.on("new peer", function(peer){ // Runs once for each smart device
9
10     peer.getStreams(function(streams){
11
12         if(streams["chat"]){ // Pipe external stream to Unix shell
13             streams["chat"].pipe(process.stdout);
14         }
15
16     });
17
18 });
```

Listing 4.3: Code example for a P2P chat client.

4.4.2 Distributed Volume Control

One of the first example prototypes, allowed other devices to control its audio volume through a property using the *amixer* command in Linux. The *amixer* command is a part of Advanced Linux Sound Architecture (ALSA) and was used in order to get and set the volume. To spawn child processes, the *child_process* module was used.

4.4.3 Video Streaming

When implementing streaming functionality in the Home-Thing module, I created two prototypes for streaming video between them. In order to access a video stream in one of the prototypes, the *child_process* module was used, as shown in Listing 4.4.

The prototype receiving the stream implemented a *new peer* listener that checked whether or not the discovered devices shared a stream named "video". If found, it was piped to the standard input stream of a child process running the media player *VideoLAN*.

```

1 var app = require("home-thing")(name, description);
2
3 // Get the child_process module
4 var terminal = require('child_process');
5
6 // Acquire camera stream
7 var stream = terminal.spawn("ffmpeg", ["-i", "/dev/video0", "-f", "avi", "-"]);
8
9 // Share the stream
10 app.addStream("video", "Video from web camera", stream.stdout);

```

Listing 4.4: Example code for recording video from a camera.

4.5 Documentation

The documentation of the Home-Thing module was created in order to support developers and provide an overview of the available features. It was written in markdown⁵ and is available in the *README.md* file in the module repository on GitHub. Figure 4.5 depicts a cutout from the documentation. The complete documentation is attached in Appendix A.

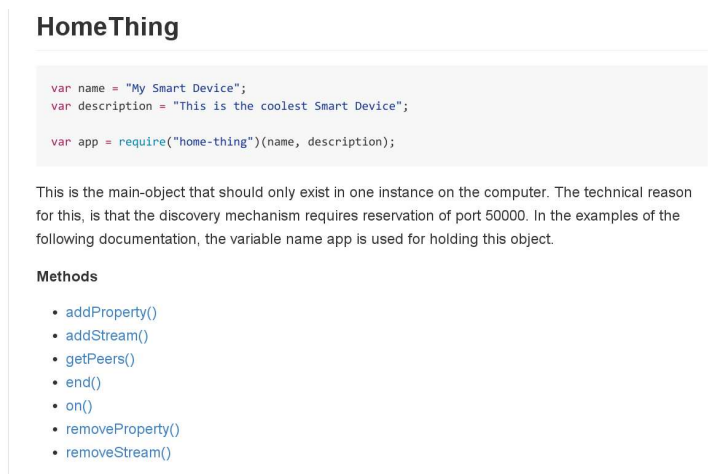


Figure 4.5: The Home-Thing module's documentation on GitHub.

⁵<http://daringfireball.net/projects/markdown/>

Chapter 5

Results

I think it's very important to have a feedback loop, where you're constantly thinking about what you've done and how you could be doing it better.

- Elon Musk

The Home-Thing module was created to serve as a proof by demonstration, allowing me to later give answers to the research questions. Furthermore, obtaining feedback from other perspectives was also important in order to provide a more sufficient discussion about the implementation choices. Therefore, students were invited to take part in a series of workshops at Østfold University College, where they developed smart devices based on the module. In order to gather feedback on the module, each of the participants shared their thoughts about the session during semi-structured interviews after the workshops. To capture important details from the the workshops and the interviews, both video and audio was recorded.

5.1 Workshops

In order to obtain sufficient feedback, three workshops were followed by interviews. Both sexes were represented among the participants, who had backgrounds from computer science, and was between 23 and 32 years of age. Three workshops were conducted, where the two first involved only participants with no prior exposure to the project. In this report, the participants are given other names to allow them to be anonymous. A more detailed overview of the participants is listed in Table 5.1.

Name	Age	Gender	Workshops
Grace	26	Female	1
Jade	25	Female	1
Brendan	32	Male	1 and 3
Ryan	23	Male	1 and 3
Lars	23	Male	2

Table 5.1: Overview of the workshop participants.

All of the three workshops had focus on smart device development. The participants were given tasks to complete in order to expose them to as much of the module's features as possible. More details about each workshop will be described in Section 5.1.5, Section 5.1.6 and Section 5.1.7.

5.1.1 Available Smart Devices

In order to aid the participants during the workshops when debugging their prototypes, a smart device monitor was developed with the intention of providing a live overview of the smart devices on the network. The smart device monitor hosted a web page that displayed representations of each running smart device. As the participants started their prototypes, circles appeared on the monitor containing the devices' names, surrounded by representations of their shared properties and streams. Green circles indicated that the resource was a property and blue circles indicated that it was a published stream. The smart device monitor's web page is depicted in Figure 5.1.

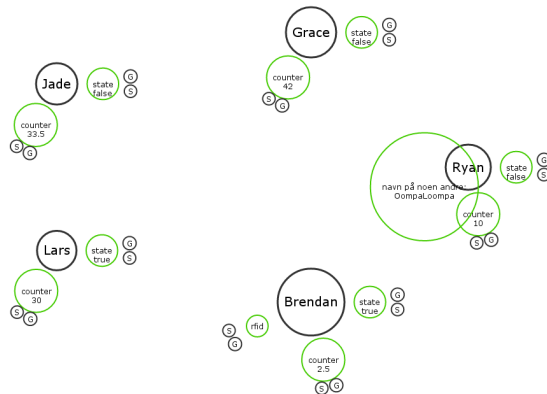


Figure 5.1: Smart devices illustrated by the smart device monitor.

When utilizing the Home-Thing module it takes two smart device instances in order to exchange any data. Developing and testing would therefore require other working smart device instances during the workshop. To avoid situations where participants were dependent on each other in order to test their prototype, I added an additional smart device that they would rely on instead. This smart device was connected to an Radio-Frequency Identification (RFID) reader in order to output its data into a shared stream. When preparing the second workshop, a property was also added to the RFID smart device. The smart device monitor's representation of the RFID smart device is depicted in Figure 5.2.



Figure 5.2: RFID smart device illustrated by the smart device monitor.

5.1.2 Tutorial

In order to remind the participants of how JavaScript differed from other languages, and more importantly, bring the less confident to a certain level, a tutorial was held in the beginning of the workshop. In order to show some examples of code to the participants, a simple web page using Bootstrap¹ and WebSockets was created. The tutorial page is illustrated in Figure 5.3. The link to the web page was given to the participants to let them study the example code on their own screens during the tutorial. To literally ensure that everybody was on the same page during the tutorial, WebSockets were utilized in order to make the navigation collaborative. Anyone displaying the tutorial were able to decide which page the participants' browsers would show.

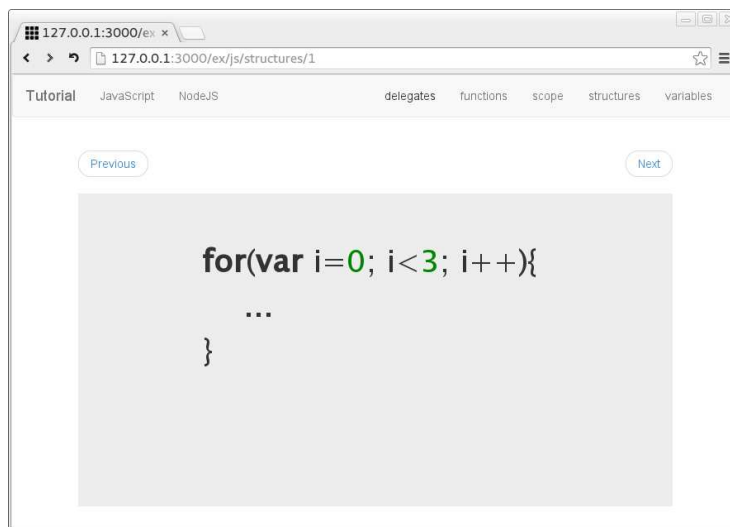


Figure 5.3: Web based tutorial page showing a code example.

As the tutorial progressed and the more advanced examples were presented, the participants were asked to speak out when they were unable to follow. When questions were asked, all the participants were given another walk through of the code with more detailed explanations. In the last part of the tutorial, Node.JS was introduced with practical assignments for each of the participants in order to ease the transition over to prototype development.

5.1.3 Observations

During the development session, I observed the participants and gave them advice when they requested it. With four participants in the first workshop, questions were frequently asked and required a lot of attention. Therefore I provided additional recording equipment instead of writing down the questions and observations. Further, by answering the participants' questions, I was able to form an impression of how each participant's work progressed during the development sessions.

¹<http://getbootstrap.com/>

5.1.4 Interviews

After having their first encounter with smart device development by using the module, each participant took part in a semi-structured interview. Therefore, the questions had to be carefully planned in order to reveal relevant thoughts about the development session and the Home-Thing module as described in Section 3.6. The reason for choosing this method was that I wanted to have richer conversations with each of the participants yielding reflections and ideas for improvement. I also wanted to be able to steer the conversation in the right direction when it digressed. The interviews that followed the first and the second workshop were individual in order to keep the participants from influencing each other.

During the interviews, the participants were first asked questions about their backgrounds as programmers with focus on Internet of Things (IoT), JavaScript and Node.JS. Following, the questions were directed more towards their progress during the development session and how they used the different features of the Home-Thing module. In the last questions, they were asked to suggest how the module could be improved based on their experience.

5.1.5 First Workshop

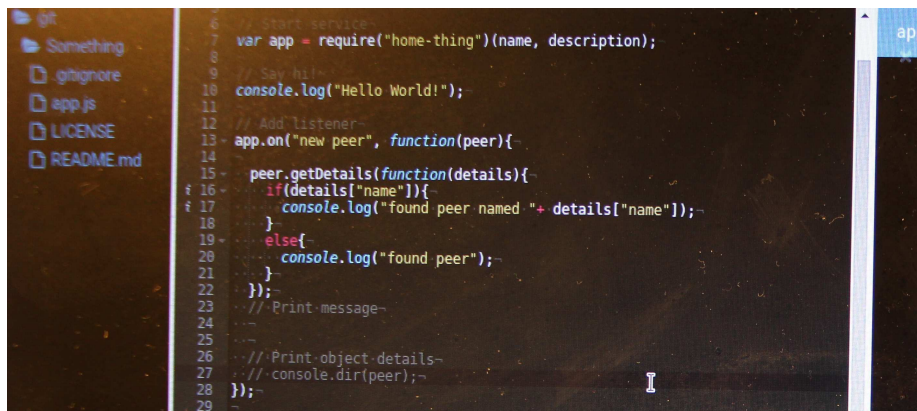
The first of the three workshops was conducted with four participants and featured approximately a thirty minutes long tutorial. The tutorial had basic JavaScript examples as well as instructions for how to use Node.JS in order for the participants to install modules and run smart device prototypes. More details about the tutorial is described in Section 5.1.2. This was followed by a session with prototype development lasting for approximately an hour. During the development session the participants were given the following tasks:

1. Change the name of the smart device prototype.
2. Implement a stream listener.
3. Implement functionality for identifying RFID-cards.
4. Explore the module further by utilizing properties on other devices.
5. Create property subscriptions.

Hardware and Development

The first workshop differed from the other workshops in the way that smart device prototypes ran on Raspberry PI computers, which was accessed remotely using Secure Shell (SSH). The participants were therefore asked beforehand to bring their own laptops, in order to connect to the smart devices. As some of the participants during the workshop expressed their concerns about developing in a terminal window, we found a solution where they instead were able to use a web based editor named Noide ². The code editor was installed as a package for Node.JS and allowed the participants to program inside their web browser instead. The Integrated Development Environment (IDE) Noide is depicted in Figure 5.4.

²<https://www.npmjs.com/package/noide>



```
6 // Start service
7 var app = require("home-thing")(name, description);
8
9 // Say hi!
10 console.log("Hello World!");
11
12 // Add listener
13 app.on("new peer", function(peer){
14
15   peer.getDetails(function(details){
16     if(details["name"]){
17       console.log("found peer named "+ details["name"]);
18     }
19     else{
20       console.log("found peer");
21     }
22   });
23   // Print message
24   ...
25   ...
26   // Print object details
27   // console.dir(peer);
28 });
29
```

Figure 5.4: Image of Noide - a Web based IDE.

Assignments

In the beginning of the development session, the participants were given a common set of goals to complete individually before the development session was over. The first goal, which was solved quickly by all the participants, was to change the name of the prototype and start it from SSH. From there, the participants were encouraged to make the prototypes listen to the stream from the RFID smart device. Then the next goal was to identify their own student card from the stream and subscribe to properties on other smart devices. Finally, in case they managed to do all this before the session was over, they would be encouraged to further explore the functionality of the Home-Thing module in order to create a smart device based on their own imagination. This provided the participants with a solid set of goals.

5.1.6 Second Workshop

Originally, the first workshop was supposed to host five participants. However, one had to reschedule and took therefore part of the second workshop as the only participant, which was conducted six days after the first workshop. Similar to the first workshop, the second workshop had a basic tutorial, a session for development and finally a semi-structured interview. From the first workshop it was reported that the Raspberry PIs were slow to develop on. Therefore, I provided the participant in the second workshop with a desktop computer to ensure better performance for development. It ran Ubuntu³ instead of Arch Linux ARM⁴, which was the environment used in the first workshop. Therefore, the user did not need to bring his own laptop as he was able to develop directly on the computer.

The assignments for the second workshop were the same as the ones listed in Section 5.1.5, and were given to the participant as the development progressed. Having only one participant, made it easier to observe the prototype development and allowed for closer follow up.

³<http://www.ubuntu.com/>

⁴<http://archlinuxarm.org/>

5.1.7 Third Workshop

The third workshop consisted of a development session and an interview. No tutorial was needed as the two participants of this workshop already had been involved in the first workshop. The purpose was therefore to allow them continue with the prototype development in order to use the remaining functionality, which they did not have time to explore during the first workshop. In the second workshop, a desktop computer had been used for hosting the smart device prototype. This had been notably faster to develop on and was therefore also used in the third workshop. As there were two participants, a second computer was also set up.

		Brendan	Ryan
Property	Read		✓
	Write	✓	✓
	Subscribe	✓	✓
Stream	Create	✓	✓
	Publish	✓	✓

Table 5.2: Overview of assignments in the third workshop.

As the participants had prior experience with the module from the first workshop, and had taken part in individual interviews, the interview that followed the third workshop was conducted with both participants present. The assignments given during the third workshop, however, were individually tailored to each of the participants, as they continued the prototype development. These were written down in text files before the workshops, and transferred to the participants' computers, allowing them to look up the tasks if they forgot them during the workshop. Table 5.2 summarizes the tasks that were given to the participants in the third workshop.

5.2 Prototype Code

In order to avoid losing valuable information, backups of the participants' prototypes were taken after each workshop. The code provided additional insight into the conversations from the interviews and was helpful when analyzing the transcriptions. As the video did not have high enough resolution to show code displayed on the screens, I looked at the backup code instead to confirm my interpretation of ambiguous statements, given by the participants.

5.3 Transcripts

During the workshops, both video and audio was recorded. There were two cameras with microphones recording from different angles and one audio recorder placed on the table next to the participants. To transcribe the material, I first created scripts in Python for copying raw video from the video cameras' memory cards, which utilized the command *ffmpeg* to added compression. Video from a session was separated into multiple files on each camera, each containing

metadata such as creation timestamps. When compressing the video, I therefore made the script extract and save the timestamps in order to allow syncing the different material later when transcribing. Following a Web application, depicted in Figure 5.5, was created allowing me to effectively speed up the transcription process.

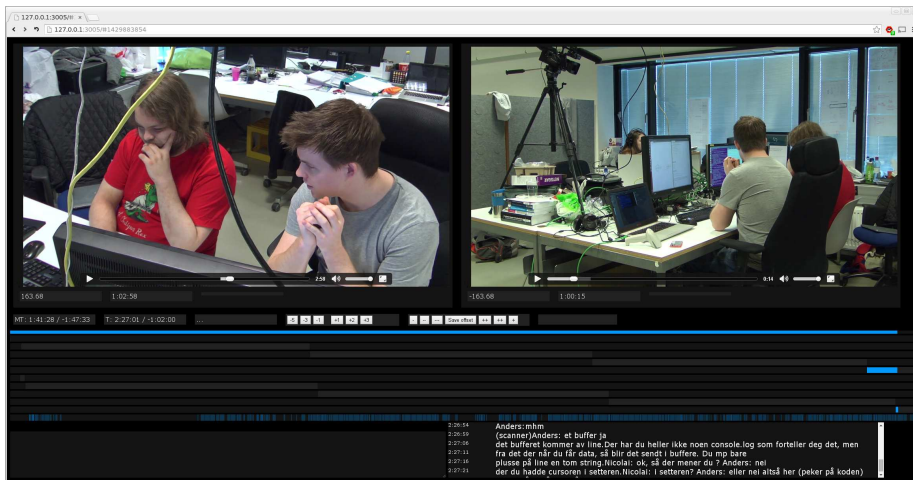


Figure 5.5: Custom solution for transcribing audio and video.

It featured a timeline for the video and audio material, two frames for displaying the video and a text box for entering the notes. When writing in the text box, a timestamp from the current position of the video was recorded and appended along with the notes to a text file on the server. Additionally, the text box featured auto-complete functionality for writing the names of the participants and myself. In the Web application keyboard shortcuts were used for navigating the video. For instance, the shortcut *Ctrl + Enter* would play 5 seconds of video and then pause it.

5.4 Analysis

The transcripts from the video and audio were analyzed in a process inspired by open coding and focused coding. Through the analysis, transcribed material was closely examined, tagged and categorized. The rest of this section describes more details about the procedure of analyzing the data. Descriptions of the methods open coding and focused coding are detailed in Section 3.7.1 and Section 3.7.2.

The first step in a coding process is to get to know the data. Having been present during all interviews and workshops, I found it easy to remember what was said by who. Additionally, the transcription tool I created, was tested with video material from the workshops. Therefore, some parts of the video material was very well known after encountering a number of bugs.

The next step was to go through the transcripts and break down the text into chunks, in order to assign tags to the content, which consisted of one or more codes. Some of the most important tags were later refined to concepts.

The relevance of the transcript contents varied, and I therefore alternated

between coding line-by-line and phrase-by-phrase. Each tag enclosed text and one or more codes with square brackets in order to stand out from rest of the text. Most of the tags were constructed and based upon meaning of the sentences, but sometimes it was better to use in-vivo codes instead, where the words used by participants serves as codes. The tags and codes made it easier to go through the data in order to compare the thoughts on specific subjects from the different participants.

The following selection from one of the transcribed interviews demonstrates how the tags were applied:

Interviewer: Do you have any prior experience with coding JavaScript?

Lars: Yes, but very basic [basic "JS" "experience"] and only client-side programming "JS" "experience" only on "frontend".

Interviewer: So, you don't have any experience with Node.JS?

Lars: No. [no "experience" with "Node JS"]

After having gone through the material several times, the analysis process was focused on grouping and linking the concepts. With patterns beginning to emerge, it made sense to categorize the concepts.

Background	Documentation
Issues	Module
Programming	Reflections
Workshop	

Table 5.3: Categories from the analysis.

With both the research questions and the induced concepts in mind, the categories were finally refined to be the ones listed in Table 5.3. The categories were used to structure the findings, detailed in Section 5.5.

5.5 Findings

In this section I present the findings from the three evaluation workshops and the interviews that followed. The structure is based on the categories that emerged during the process of analyzing the data with open coding and focused coding. An overview of the categories is given in Table 5.3. Each category section lists its corresponding concepts and details the relevant findings.

5.5.1 Developers' Backgrounds

During the semi-structured interviews, the first questions focused on prior knowledge and experience with relevant programming and IoT. The category *Background* therefore revealed itself as an important category in order to group the concepts in Table 5.4.

From the interviews it could be gathered that all the participants had prior experience with frontend JavaScript. However, their confidence varied. While

Backend	Experience	Frontend
IoT	JQuery	JS
Node JS		

Table 5.4: Concepts under the category of *Background*.

some of the participants reported to have a lot of experience with JavaScript, others expressed that their only encounter with it was one time when they tried out some example code on a Web page.

Three of the participants from the first workshop reported to have assisted in teaching other students basic programming using JavaScript. The same participants did also have experience with the programming language from earlier projects related to their own studies. A participant from the first workshop proclaimed the following during her interview:

Jade: I do have quite a lot experience with JavaScript and I used D3 in my Master’s Thesis.

Another participant said he had only used JavaScript once before. Still, he had a lot of experience from programming other languages with syntactical similarities, such as C, C++, Java and C#. Taking part of both the first and the second workshop, this participant turned out to progress the most, and was able to complete all the tasks given to him. Nonetheless, server-side programming using JavaScript was new to each participant, except for one, who reported to have studied some example code for Node.JS where he created a chat-client. However, he emphasized that this was months ago and he had not put much effort into it.

When asked to tell about their knowledge of IoT, all the participants could say they had heard about the concept before. Moreover, two of the participants also had experience working on a related front-end project for controlling smart devices. Another participant who had heard about IoT, but seemed uncertain about what it actually implied, explained:

Lars: I’ve heard about the internet of things, but have never read up on it. I’ve just heard that it has something to do with devices being connected to the internet.

The remaining participants stated that they did not have any practical experience, although one of them mentioned that he was interested in the subject and had thought about experimenting with the technology.

Both basic JavaScript and IoT was covered during the tutorials held in the beginning of both the first and the second workshops. Having a main focus on JavaScript, the tutorial showed the participants example code with explanations about how and why the results would appear as described. During the tutorial in the first workshop, one of the participants remarked that he thought JavaScript was a very flexible language, as it allowed for code to be written in many different ways. Based on observations, questions and comments from the participants, during the tutorial, they all seemed to discover new things about JavaScript, often related to syntax.

5.5.2 Workshop

The category *Workshop* derived from three workshop related concepts, which are detailed in Table 5.5.

Assignment Help Question

Table 5.5: Concepts under the category of *Workshop*.

In all the workshops, participants were given a number of tasks to complete. During the first workshop, these were presented verbally to the participants as a list of goals, and were encouraged to complete these before the session was over. None of the participants did manage to complete all the tasks during the first and second workshop.

Despite being unable to reach all his goals, the developer from the second workshop progressed most of all the participants. Being the only participant he had the advantage of getting help whenever he asked for it. Another participant explained during her interview:

Grace: During the workshop I forgot what the task was. It was fine, but I've should have written it down somewhere.

In the third workshop, the participants were therefore given written down assignments. As the participants of the third workshop both had taken part in the first workshop, their tasks were made individual and aimed at encouraging them to study and utilize the remaining features of the module. One of the participants, given five tasks to complete, first expressed doubt about being able to complete all comprehensive tasks before the workshop ended. When later asked about his thoughts about the assignments, he remarked that it would be better to have tasks with more focus on practical use. He also suggested that it could have been more fun to do the tasks if it was easier to get hold of the required smart devices. The other participant sat intently concentrated with his code and exclaimed in the end of the workshop:

Brendan: I can't stop before I get it to work! It is still false!
(Talking about code)

When he was finally able to fix the error, he exhaled with relief saying that he felt joy and was proud of the work he had accomplished. He also reported that he had managed to complete all the tasks he was given.

5.5.3 Prototype Development

The category *Programming* was based on concepts linked to the development process, in which the participants created smart device prototypes. The concepts for this category is detailed in Table 5.6.

Observation Code comprehension Code placement

Table 5.6: Concepts under the category of *Programming*.

JavaScript as a programming language was described by the participants as being both accepting and user friendly. As described earlier in Section 5.5.1,

some of the participants had more experience with the language than the others. This soon became a topic of interest among the participants during the first workshop. While later being interviewed, one of the participants explained that he actually had fun showing another participant with less JavaScript specific experience, how accepting the language was towards typos in the syntax. For instance, he had demonstrated how semicolons could be removed from the code without causing errors. The other participant, observing the demonstration, later reported the following during his interview:

Brendan: I managed to do the things I set out to do, although I haven't got much experience with JavaScript. It was a bit chaotic in my opinion, compared to the other languages I know. However, the code was understandable and I figured out where to place it as well.

He also added that he had just recently encountered delegates for the first time when writing code in C#, which he found to be similar with the function expressions in JavaScript.

What appeared to be the most challenging part for the participants when developing prototypes, was to apprehend and imagine the execution timeline for the different blocks of code. The usage of function expressions during the workshop seemed to confuse most of the participants. More details about this particular issue will be described in Section 5.5.6.

Remarks and questions about where to put code was most apparent during the first workshop. The participant who appeared as the most confident JavaScript programmer, and the participant who had almost no experience with the programming language, both asked questions about where to put the code during the first workshop. The other participants did not, but reported later during the interviews, that they had observed the other participants when programming. The common achievements among the participants during the three workshops can be summarized as:

- Modification of prototype names.
- Utilization of "new peer" listeners.
- Filtering of specific smart device objects.
- Reading external streams of other smart devices.

In addition, some of the participants also created properties, changed external property values belonging to other smart devices, published streams and created subscriptions. The functionality of the module that was utilized the least, was the property subscription feature. This feature was used in the monitor described in Section 5.5.4 and proved successful, but was only tested by two of the participants during the third workshop.

5.5.4 Documentation

During the workshops and the interviews, the topic of documentation was brought up by the participants multiple times. After having analyzed the transcripts and found documentation to be one of the key concepts, it also became

Documentation Example code Help

Table 5.7: Concepts under the category of *Documentation*.

clear that it would also fit as a category. The concepts placed under *Documentation* is detailed in Table 5.7.

Before utilizing the module, the participants were given links to the documentation, which is described in Section 4.5. The participants all reported having used some of the example code from the documentation. When sharing his thoughts regarding the prototype development, Lars, the participant from the second workshop explained:

Lars: When learning new things, I like to sit alone, reading the documentation and just play around with example code. I'll turn things off and on and look at how it affects the functionality. On the spot programming comes to mind when I think about today. However, I was able to ask questions and was also given some space every now and then. I really think it was a good way of working, and when problems occurred, I could just ask about it.

The second workshop was unlike the other two. It had only one participant and a more quiet atmosphere. Lars also had the advantage of having help available whenever it was needed. Still, he was for the most part determined to solve the problems himself by reading the documentation.

Two of the participants from the first workshop, Jade and Ryan, reported during the interviews that they were not satisfied with the depth of the documentation. Jade said she would have become frustrated if she was unable to ask for help when encountering problems using the module. She elaborated that the descriptions in the documentation did not provide a sufficient level of details for her use. Ryan focused more on the example code when talking about the documentation and explained:

Ryan: The examples in the documentation were good. Although there ere given some examples of practical use, I wish there had been more, in order to give me a helping hand with for instance streams. Where and why should I make smart devices stream? I don't get it. I need more... I would prefer to have a working example in there that could be copied, pasted and manipulated in order to show why it worked.

During the interview, Ryan also explained that he would have liked to have some more help in the documentation for code that was not a part of the module as well. He elaborated that documentation and example code for how to create a writable stream for instance, would have been helpful.

During the second and the third workshop, creating an empty writable stream for logging messages turned out to be more difficult than expected. What the participants had to first create a writable stream that appended log messages to a file. This file was then read using the *child_process* module for Node.JS in order to run the Unix shell command *tail -f*, which would continuously check for new content in the end of the file and print it. By doing this,

they were able to produce and publish streams for sharing log messages with other smart devices.

The participants liked to have example code available in the documentation. The participant who progressed the most during the first workshop, Brendan, stated that when using the documentation, in most cases, he would only look at the example code. He also added that he preferred to ask for help instead of looking through the documentation. Still, in multiple occasions when asking questions, he was not given a verbal explanation, but referred to the right place in the documentation.

5.5.5 Home-Thing Module

When creating the categories, a number of the concepts emerged as clearly linked to the module. Therefore, *Module* became a category consisting of the concepts described in Table 5.8.

Module	Module bug	Object oriented
Property	Stream	Structure
Subscription	Example of use	

Table 5.8: Concepts under the category of *Module*.

The participants gave positive feedback about their experience of using the Home-Thing module, when creating smart device prototypes. Brendan, who had the least experience with JavaScript, reported the following:

Brendan: It was plain and simple to use the module, really. Adding properties to the prototype and firing up an instance was easy, having you there to help me, although I could have figured out more by myself. I think it was better to get help from you, though.

While some of the participants were not shy to ask for help, others only requested help a couple times when developing prototypes. The participants who during the first workshop asked most frequent for help, also managed to complete more of the tasks and showed a more wide use of the module's features.

When the Home-Thing module was developed, a set of tests were created in order to find and fix bugs early on. Although many were found and patched, there was one bug that revealed itself during the second workshop. It caused the module to crash when string values from external properties were parsed. In the interview that followed, the participant from the second workshop elaborated:

Lars: Of course, one feels lost when something is wrong with an encapsulated system such as the module. However, I just accepted that something was wrong, and then it was fine.

The bug turned out to only affect string properties. Therefore the problem was avoided by using an object property, containing a string value. This bug was fixed before conducting the third workshop.

In the first and second workshop none of the developers managed to complete all their tasks before the development session was over. When interviewed after the first workshop, Ryan reported that his Raspberry PI had been running

slowly. He also explained that this restricted his goals when developing. However, he was able to continue in the third workshop and could finally make his prototype read both a property and a stream from another smart device. When discussing streams and properties in the interview after the third workshop Ryan reported:

Ryan: My impression is that it was simpler to quickly get results using properties compared to streams. Still, I think that streams can yield better results with respect to how much time you spend on it. One can also get more interesting functionality.

The other participant from the third workshop, Brendan, was able to read an external stream already during the first workshop. His impression was that streams were easy to use. This view was also shared by Lars from the second workshop, but not by the two remaining participants Grace and Jade. Jade, in particular was frequently requesting help when developing the smart device prototype, in order to understand streams properly. When later asked about her experiences with streams, she answered:

Jade: I didn't understand how to use streams. I haven't used it except in Java when handling file streams. It seemed like the same concept, but I didn't understand it completely, so that was the main problem.

As mentioned in Section 5.5.3, the subscription feature for properties was overlooked both during the first and second workshop. In the third workshop the assignments were given to Ryan and Brendan individually with different goals, which explicitly required use of the subscription feature. Although they both managed to subscribe to an external property, the interview afterwards did not yield many thoughts about it. The code written by the participants later revealed that only one of the prototypes were able to subscribe to the right property. The other prototype used the wrong property name and did therefore not work. Everything else was set up right.

By offering the developers help during the workshops, it gave insight into how they perceived some of the features in the module. An interesting point raised by one of the participants during the third workshop, was:

Brendan: How do streams differ from subscriptions? A stream is in a way a subscription, right? I do after all get the information.

The documentation did not feature explicit details about how the module worked behind the scene, which was done intentionally, to let them use the functionality without having to read up on the details behind them.

When asked about their thoughts regarding the Application Programming Interface (API) structure, the participants responded that they were pleased. However, some of them added that it was difficult to give a reliable answer, as they did not have anything else to compare it with. When touching in on the topic of structure during the third workshop, Ryan and Brendan disagreed on how to improve the structure:

Ryan: When I understood how things were structured and that I needed to iterate the objects' children, it was fine. [...] Let's say you

want to connect two lamps. A lot of code has been written in order to get hold of both lamps, however, when you connect these together it's done in an instance, because there is almost no code required for that part. I think if you were able to tone down the overhead before you come to the juicy bits, it would help a lot. Then, the tasks would probably have become more amusing as well. [...]

Brendan: That's probably all right. However, it depends on how often you need to obtain the objects. If you only need to retrieve them once in your code and have access to the variables all the places necessary, it's not that inconvenient. However, if you needed to initiate retrieval of a stream every time, that's when it would be inconvenient.

Furthermore, it was emphasized by Lars from the second workshop, that he liked how the module object was used, and how he could refer to it anywhere in the code. He also added that he liked the object oriented approach of the module, as he was used to the object oriented way of thinking and felt confident when using it.

5.5.6 Issues

The participants all encountered problems during the workshops. This became visible through the concepts, which were grouped under the category of *Issues*. The concepts are detailed in Table 5.9.

Disorderly code	Indentation	Monitor
Problems	Slow	Syntax

Table 5.9: Concepts under the category of *Issues*.

Before the workshops, a tool for helping the developers visualize the available smart devices was created. The monitor, is described in Section 5.1.1. In the first workshop its overview of the smart devices was displayed on a large screen in order to be visible to each of the participants. In the workshops that followed, it was displayed in local browser windows on the participants' desktop computers. About the monitor, one of the participants, Lars, remarked:

Lars: I liked the fact that I could see if my application worked by starting it and watch it appear on the monitor. I also liked that I could see the monitor from my application.

The monitor was created as a smart device based on the Home-Thing module and was visible to the other smart devices in the same way as any other smart device. A screen displaying the monitor Web page is depicted in Figure 5.6.

In order to display property values, the monitor was programmed to subscribe to all available properties on the smart devices that it discovered. This feature was put to the test during the first workshop, when Ryan decided to create a smart device with a large amount of properties in order to make the monitor visually draw a full circle of properties. Soon after, he also wanted to see what would happen if he updated their values every millisecond. After having found out, Ryan exclaimed:



Figure 5.6: Unforeseen use of the monitor.

Ryan: I'm not sure what to do now. The monitor has crashed by the way. I think it has something to do with that 1 millisecond interval thing.

The result of this experiment was that the monitor stopped responding. It is difficult to determine which process stopped responding first. However, what became clear, was that both the browser and the monitor process running the smart device instance had to be killed by force and restarted.

Another issue with the monitor, was displaying only the available devices when the participants' instances crashed. Therefore, the crashing prototypes' representations accumulated on the monitor display and became confusing for the participants. During the workshops, restarts of the monitor therefore became a frequent request. In one of the interviews after the first workshop, Grace mentioned having had problems with her Raspberry PI computer:

Grace: The Raspberry PI was a bit slow, but it became even worse when the other participants did weird things.

Interviewer: Was it slowed down?

Grace: Yes, I think so. When the Raspberry PI that belonged to Brendan had problems it took a long time before any text in the SSH window appeared.

Interviewer: Did you notice anything when Ryan added hundred properties as well?

Grace: I'm not sure.

In addition to slow hardware, the participants also encountered issues regarding the programming itself. The way curly brackets and function expressions were used in the example code confused some of the participants. In the third workshop, Ryan experienced losing track of how his code worked due to inconsistent indentation, which increased its complexity each time he added code.

Another issue was related to the participants' understanding of when the function expressions would run. These were triggered by events, and were executed asynchronously. Based on observations during the workshop, this seemed to confuse the majority of the participants when developing and testing their prototypes. For instance, when using of the function `getPeers()`, participants were unsure of what happened when they did not get any objects returned. What the function does is to return all available peers, which was indeed what the participants wanted. However, when placing the function call sequentially with the initialization call for the module, it would not return any peer objects. The reason was that it took a certain amount of time, approximately a second, before the Home-Thing module was able to discover peers. Meanwhile, the code that was placed right after the initialization call, executed before the new peer event was triggered in the network thread by a response packet.

5.5.7 Reflection

Reflecting thoughts from the participants became a common theme in the transcripts. The category *Reflection*, was therefore created to capture the ideas of the participants regarding how the module could be improved. The associated concepts to the category are detailed in Table 5.10.

Communication	Difficult	Easy
Reflection	Suggestion	

Table 5.10: Concepts under the category of *Reflection*.

During the interviews, Jade, Lars and Brendan shared their thoughts about the communication between the different smart devices. Lars described it as being easy to achieve, although it took some time to understand how it could be used. Jade however, emphasized that she had problems understanding how the module connected to other instances behind the scene. She wanted to understand more about how it worked, and not only that it did. Brendan also wanted to know more about how the module worked, as quoted in Section 5.5.5, where he mention his thoughts regarding the difference between streams and subscriptions. However, not all participants felt it was necessary to know such details. When asked about her accomplishments during the workshop, Grace answered:

Grace: It went just fine. I usually don't think so much about how things work in the background. Instead, It's better to just have a template for how to do things when programming.

Moreover, she reported to have been able to retrieve the right smart device, and read its stream.

During the interviews, the participants expressed their thoughts about what was easy and what was difficult when developing with the Home-Thing module. Jade thought streams was difficult to understand, as quoted in Section 5.5.5. She did manage to print out the contents of the stream. However, she also said that it was difficult to understand why it worked. Another thing that thing that was difficult, Ryan said, was to distinguish between code that was part of the Home-Thing module, and code that was not. It was not clear to him that the

code used during the third workshop, for creating streams sharing log messages, was not a part of the module. Except for this, he said it was easy to understand much of the code right away.

When the participants were asked to suggest improvements for the module, Ryan in particular had many thoughts to share. The following list contains a selection of the participants' ideas for improvements:

- By exposing the peer objects in a global variable, he imagined it could be easier to iterate the various smart devices and their properties. This would differ from the current situation, where instead, the peer objects are returned by a method.
- Flatten the structure of peer objects by removing the *getDetails()* function, and make the name and description available as properties.
- The property type named *text* should be renamed to *string*.

It was also elaborated that the target audience for the module, being developers, would understand what a string was. Therefore it would be unnecessary to call it anything else.

Other more general improvements suggested by the participants was to add additional example code and descriptions to the documentation, which is further explained in Section 5.5.4.

5.5.8 Summary

The participants had no or little prior experience with server side JavaScript programming and Node.JS. However, they were for the most part familiar with JavaScript and client side development. The IoT was something they all had heard of before, but something that most had limited knowledge about.

The participants made different progress during the workshops when developing prototypes. In their first session, none of the participants were able to complete all the assignments given to them. Judging by the feedback from the interviews, this could have been caused both by time constraints and that the assignments were given verbally. Also, the enthusiasm for doing the given assignments varied between the developers.

Although the participants all managed to develop functionality for their prototypes, there was confusion regarding when the function expressions would be executed. The common achievements of the participants when developing prototypes during their first workshops included filtering of smart devices using names and reading streams. The subscription feature of the module was only utilized by two of the prototypes.

With regard to the Home-Thing module's documentation, all the participants used it, but some wanted more depth and descriptions of the functionality behind the API. It was also requested more example code and documentation for code which was not directly a part of the module.

The module was positively received by the participants. Some participants asked questions more frequently than others when they needed help. Other participants found clues to how they could solve problems by looking at code produced by other participants. The ones who asked more frequently for help did also progress most during the first workshop. During the second workshop, a bug

in the module was discovered. This was resolved quickly with a workaround, and it was fixed before the third workshop. The streaming feature, was something that the participants said took time to to understand. It was also unclear how streams differed from subscriptions, which was not explicitly described in the documentation. With regard to the structure of the module's API, the participants found it difficult to suggest improvements, as they did not have anything to compare it with. However, it was suggested to tone down the use of asynchronous calls and further decrease the depth of the structure.

The experimentation of one of the participants, froze the monitor process during the first workshop. Additionally, the monitor accumulated representations of devices that was not able to shut down gracefully when crashing. Other issues reported by the participants was related to slow hardware and code containing asynchronous methods that did not function as they first expected.

The participants thought that communication between smart devices was easy to achieve, although it was difficult to understand how it worked behind the scene. It was also reported that it was difficult to identify code was a part of the module and code was not. When asked for possible improvements for the module, one of the suggestions was to make the smart device name on remote peers available as a property. Another suggestions was to rename the property type *text* to *string*, and to add more example code in the documentation.

In Chapter 6, I will discuss the research questions, the Home-Thing module and the findings from this section.

Chapter 6

Discussion

Dreams feel real while we're in them. It's only when we wake up that we realize something was actually strange.

- Cobb

In this chapter the research questions will be discussed in light of the project's exploration phase and the findings from the evaluation phase. The research questions are as follows:

RQ 1: How can smart devices discover and connect with similar peers on a local network without depending on a common server?

RQ 2: How can smart devices in home networks exchange data, send notifications and affect each other by utilizing proven Web technology for transferring states?

RQ 3: How can data streams be utilized by smart devices in order to transfer continuously updated data from sensors?

In Section 6.5, Section 6.6 and Section 6.7, each of the research questions will be discussed individually. They will first be given an introduction with details about the motivation behind them, before describing the final solution. This is followed by findings from the exploration phase, including feedback given by the participants that took part in the workshops. To avoid repetition, I will first address topics that are common to the research questions in the following sections.

6.1 Project Structure

The goal of exploration is to gain insight into an area and find challenges that need further investigation [29]. In this project, I combined exploration with qualitative evaluation, as detailed in Section 3.1. This was done in order to first find a challenge to overcome, create a possible solution and then evaluate the solution with developers, giving feedback from other perspectives.

During the evaluation workshops, both audio and view was recorded. This allowed for later transcribing and analyzing the developers thoughts and questions. The developers were given help when asking questions during the workshops. To be in the role of an observer and at the same time offer expert help to the participants was challenging. Additionally, I had to administrate the devices on the network in order to restart the monitor several times as later discussed in Section 6.5.2. The video and audio recordings later helped me uncovered details, that was missed during the workshop due to being busy. In similar evaluations, screen recording may prove helpful in order to later be able to watch and analyze how the work of the developers progress.

6.2 Development Platform

In this project, Node.JS was chosen to be the implementation platform, due to its rich collection of modules, created by the community [56]. Additionally, it is able to handle a lot of concurrency [56], which is a good quality to give smart devices, allowing for efficient network communication. The programming language used in Node.JS, JavaScript has over the years become more popular [3] and is now the most active language¹ on GitHub². GitHub is a Web based git repository, where you today can find the majority of Node.JS modules [44]. More details about Node.JS can be found in Section 2.4.

JavaScript was described as both being an accepting and a friendly language by the workshop developers. According to Flanagan [23], JavaScript first appears as a simple language. It has features that makes it forgiving such as optional use of semicolons. However, it is a full-featured programming language, which can be frustrating to use for non-trivial tasks, without having a solid comprehension of it.

6.3 Application Programming Interface

After having studied Internet of Things (IoT), a Node.JS module was created for abstracting service discovery and data exchange from smart devices. This separated the underlying communication functionality and the other smart device code, allowing communication experts to handle security and privacy mechanisms in later versions of the module.

Creating the prototype and deciding its Application Programming Interface (API) structure myself, was something that first emerged as the best solution after having considered some alternatives. In the Controlling the Home project, which is described in Section 1.1.1, Participatory Design (PD) was used in order to design a mobile interface for controlling smart devices. Having positive experience with this technique, it was initially considered as a possible approach for defining the prototype API. Another possible solution, was to create a basic prototype and improving it over time, by inviting developers to work on the project on its GitHub repository³. It was possible that this approach would result in a large number of deriving projects, yielding interesting results.

¹<http://github.info/>

²<https://github.com/>

³<https://github.com/anskaal/home-thing>

When creating the module, its API was designed for expert users of the Node.JS platform, as described in Section 6.2. Conducting a PD workshop in order to define an API would require the participants to have both prior experience with IoT and a certain level of expertise in JavaScript and Node.JS if it were to make sense for the targeted developers. Inviting developers to the GitHub repository and starting a collaborative effort, in order to define the API, could take months without a guarantee of yielding good results.

Being a developer with years of experience with JavaScript and having just studied IoT, I instead decided to create the API myself and later inviting developers to evaluate it. Additionally, I also developed a number of prototypes, that used the API, in order to view the API from smart device developers' perspectives. This allowed me to test various solutions and decide which methods that would be necessary.

By using the API, the smart devices based on the module were able to communicate with other smart devices through representational objects. The resulting feedback on the API structure was positive. Most of the developers said they liked to use it and one the described it like this:

Brendan: It was plain and simple to use the module, really. Adding properties to the prototype and firing up an instance was easy, having you there to help me, although I could have figured out more by myself. I think it was better to get help from you, though.

It might be argued that helping to the developers during the workshops caused an unrealistic learning situation, that would not be representative for the real world, when developers only have the documentation available. However, the reason for conducting the workshops was not to test the developers. It was to evaluate the module and get feedback from other perspectives.

6.4 Documentation

The documentation for the module API was intentionally created to only detail directly relevant information about functions, and was supplemented with snippets of example code. It was used by each of the developers during the workshop, although the usage varied, as described in Section 5.5.4. Some of the developers requested more depth in the documentation, as they wanted more examples and to know more about the underlying functionality of the Home-Thing module. Having more example code and information about when events occur, in addition to how the various methods differed from each other, could be helpful for the developers.

It was also requested more detailed information about relevant code that was not a part of the module, such as code for creating empty streams. This can be useful in order to help developers quickly find necessary code for features that are often used in smart devices. An example of such a feature could for instance be to create a log stream for the smart device that is available to other smart devices. The code for creating an empty stream is not directly a part of the module, but is still relevant to the development.

6.5 RQ1 - Discovery

In the first research question, I asked how discovery can be used for connecting smart devices without depending on a common server. With a distributed discovery process, smart devices would be functional, even without requiring users to invest in additional equipment and servers. It would also be more robust to failing infrastructure, such as single point failures in centralized servers. The smart devices was assumed to have already been configured with either static or dynamic IP addresses, as delegation of IP addresses was outside the project's scope.

In order for smart device instances to connect with the other similar devices, they must first find IP addresses to each of them. Relying on end-users to first configure the smart devices manually, would not be seen as a satisfactory solution. Homeowners should not be expected to have advanced knowledge about their home networks [42]. Thus, the Home-Thing module would have to discover the smart devices on the network itself and retrieve their IP addresses automatically.

By broadcasting User Datagram Protocol (UDP) packets, the Home-Thing module was able to announce its presence to the network to the other smart devices. The UDP packets contain an object which is formatted in JavaScript Object Notation (JSON), having one URL for each web interface. When performing service discovery, the module also listens to incoming broadcast packets and responds with its own URLs. This makes it possible to exchange links both ways, pointing to Web resources having the names of the smart devices and textual descriptions. Additionally, the resources contain the Universally unique identifiers (UUIDs) of the instance and other important values, such as minimum interval for subscription notifications. The UUID is an important value used for allowing the Home-Thing module to distinguish between different smart devices.

From the perspective of smart device developers, each smart device was made reachable through the API as representational objects. The name, description and unique identifier was made available through the function *getDetails()* in each representational object. One of the suggestions proposed by the developers was to create a global variable containing the smart device representations to allow for easier iteration. Property and stream representations are discussed in Section 6.6 and Section 6.7.

The prototype developers from the workshops successfully used representational objects without experiencing connectivity related issues. However, when module instances crashed, they were unable to notify other smart devices about exiting, which could cause problems for instances relying on them. It could for instance be a valve, controlled by a smart device subscribing to water level values measured by another device for regulating the water flow in a hot tub. A possible solution could be to implement a time-lease mechanism as discussed in Section 6.5.2.

The non-blocking IO of Node.JS was not covered in the tutorial and seemed to confuse some of the developers. It therefore had to be explained during the workshops, when they encountered issues.

6.5.1 Exploration

To allow for widespread deployment it is important to minimize the need for human intervention, when introducing smart devices to a network. Therefore, in environments with a large number of smart devices, it is desirable to have these interconnecting by themselves [11].

During the exploration phase, multiple alternatives for service discovery was found. However, most of them relied on common servers to find services, which made them unfit due to the constraint of the first research question.

A possible solution for service discovery, was to use the Node.JS module *mdns*⁴. This module was able to do service discovery through the Multicast DNS (mDNS) protocol. mDNS is further described in Section 2.5.4. To be used on Linux, however, the *mdns* module would need an additional compatibility library. The installation of this library on the different Linux distributions could complicate the development.

Prior to the project, I had some experience with broadcasting. The protocols for service discovery and the theory behind them, was a completely new field for me to study. The comprehensive job of finding fitting protocols and compare their strengths and weaknesses was becoming time consuming. Additionally, I was more motivated to create my own solution for service discovery, which I estimated to take less time than learning, implementing and debugging the alternatives. Therefore, I decided to create an implementation that was as simple as possible. By using one of the core modules in Node.JS, *dgram* to broadcast UDP packets, the Home-Thing module was able to announce its presence on the network to other listening devices. This approach did not require any extra dependency libraries outside the Node.JS platform. However, in contrast to mDNS, it limited the discovery to a single subnet [61]. In homes with multiple routers, additional workarounds will therefore be required in order to facilitate intercommunication between the subnets.

In a network with multiple devices, rules and interactions would cause hazardous situations if relying on ambiguous identifiers. Miorandi et al. [39] suggests that each smart device should be identifiable. This was possible to achieve by either returning a unique identifier that could be looked up in a database, or by enabling the smart device to provide a identifier by itself. The service discovery protocol for Bluetooth (SDP) use UUIDs in order to avoid mixing up devices [61]. To ensure that each instance was distinguishable, I added a module for creating unique identifiers within the Home-Thing module, based on the RFC 4122 [35] specification. Each UUID was shared through the web interface described in Section 6.6 and generated locally by the *node-uuid*⁵ module in the initialization phase of each module instance. The *node-uuid* module required no central registration process, in order to produce the UUIDs [35]. This way, the network of smart devices could be disconnected from the Internet and still function properly. Technical details about UUIDs are described in Section 4.2.1.

6.5.2 Feedback

The smart device developers retrieved available smart device representations by creating listeners for the *new peer* event. No technical issues related to service

⁴<https://www.npmjs.com/package/mdns>

⁵<https://www.npmjs.com/package/node-uuid>

discovery was encountered by the developers during the workshop. A possible explanation for this may be that they tested smart device development under ideal network conditions. The solution could in the future be tested in mesh networks and networks suffering from packet loss, in order to find out how the custom service discovery solution compares to for instance mDNS. Networks using the IPv6 over Low power Wireless Personal Area Networks (6LoWPAN) standard, which is mentioned in Section 2.1, may be one environment that could behave differently. It is difficult to say how home networks will develop with regard to mesh networks in the future. As the standardization efforts continue to progress in the IoT domain, it is possible that such networks will be more used. Therefore it is difficult to say whether or not smart devices based on the module can be successfully deployed in most future homes. Nonetheless, the Home-Thing module has proved that its service discovery works under the tested conditions.

Time Based Leasing Mechanisms

The monitor, described in Section 5.1.1, visualized the properties and streams on the different smart devices in the network. The developers liked having the monitor available to help them visualize the network. One of them explained:

Lars: I liked the fact that I could see if my application worked by starting it and watch it appear on the monitor. I also liked that I could see the monitor from my application.

However, during the workshops, the monitor started accumulating representations of prototype instances and had to be restarted. The module instances that crashed were unable to broadcast disconnect packets to the network. As described in Section 5.5.6, this confused the developers, which multiple times requested restarts of the monitor, in order to clear the screen of dead representations. More importantly, this suggested that in networks with unstable module based smart devices, there would be dead representations in the other smart devices, when one of them crashed. This could potentially cause unwanted behavior, as smart devices depending on the crashed instances may wait on subscription updates for instance. Zhu et al. [61] warned about this as being a potential problem in traditional distributed systems. Therefore, many service discovery protocols use time-based leasing mechanisms to address dynamic conditions. This way, the ghost representations on the monitor would be removed when the device failed to renew its lease. When mobile phones or wearable smart devices disconnects from wireless networks, it could be difficult to trigger a disconnect packet, when for instance the disconnect is caused by connection loss. Time-lease mechanisms could therefore represent a valuable feature for the future versions of the module.

Non-blocking IO

The API offered two alternative methods for retrieving smart device representations. One of the methods returned all current representations and the other allowed a listener to be defined for the *new peer* event, which run once for each discovered smart device. However, the current abstraction level of

the module in conjunction with the non-blocking IO of Node.JS seemed to confuse the developers. As described in Section 5.5.6, the code defined after the constructor call for the module, was allowed to run before any smart devices were found. The discovery process was running in parallel as it depends on other smart devices on the network to answer. A possible explanation for why this confused the developers, was that they did not have any prior experience with Node.JS. The confusion might not represent a significant problem for experienced Node.JS developers. Still, it could be worth mentioning in the documentation near the `getPeers()` function.

Global Variable

One of the suggestions for structural change in the API was to expose the representational objects for smart devices in a global variable to make iteration easier. This differs slightly from the current solution, where the objects are retrieved from a function in the API. The two different approaches are illustrated in Listing 6.1 and Listing 6.2.

```
1  ...
2
3  // Start the Home-Thing module
4  var app = require("home-thing")(name, description);
5
6  // Current solution for retrieving all discovered peers in a list
7  var allSmartDevices = app.getPeers();
8
9  // Iterate
10 for(var key in allSmartDevices){
11
12     // Print object details
13     console.dir(allSmartDevices[key]);
14
15 }
```

Listing 6.1: Example for receiving smart devices from the module today.

```
1  ...
2
3  // Start the Home-Thing module
4  var app = require("home-thing")(name, description);
5
6  // Iterate
7  for(var key in globalVarWithAllSmartDevices){
8
9     // Print object details
10     console.dir(globalVarWithAllSmartDevices[key]);
11
12 }
```

Listing 6.2: Suggested way of retrieving smart devices.

When first presented to the idea, I believed that it would not be feasible, unless the smart device developers defined the global variable themselves. Based on my understanding of JavaScript prior to this project, a global variable had to be created in the global scope before it was possible for the module to affect it from its local scope. In the book *Smashing Node.js: JavaScript Everywhere*, Rauch writes:

Variables defined within a scope are accessible only to that scope and inner scopes (that is, scopes defined within that scope) [44, p. 33].

However, as an explanation on Stack Overflow⁶ detailed, declaring variables without the *var* keyword in JavaScript can cause it to be declared globally from the local scope. In this case, when JavaScript try to resolve the variable and find which scope it belongs to, it traverses up the chain of scopes until it finds its declaration. That is, any variable with the same name. However, if it is not able to find it before reaching the global scope, the variable is then declared in the global scope [23].

As mentioned in Section 2.4, the Node.JS platform is built upon the *v8* JavaScript engine created by Google. With more evidence needed in order to confirm the validity of the Stack Overflow explanation, I went to the source code repository for *v8*⁷ and started searching for code that confirmed the explanation. This is some of the reasons I like open source code. From what I can tell, the code in Listing 6.3 shows the function responsible for the scope traversing.

```

263 ...
264 bool Scope::Analyze(ParseInfo* info) {
265     DCHECK(info->function() != NULL);
266     DCHECK(info->scope() == NULL);
267     Scope* scope = info->function()->scope();
268     Scope* top = scope;
269     // Traverse the scope tree up to the first unresolved scope or the global
270     // scope and start scope resolution and variable allocation from that scope.
271     while (!top->is_script_scope() &&
272           !top->outer_scope()->already_resolved()) {
273         top = top->outer_scope();
274     }
275     // Allocate the variables.
276     {
277         AstNodeFactory ast_node_factory(info->ast_value_factory());
278         if (!top->AllocateVariables(info, &ast_node_factory)) {
279             DCHECK(top->pending_error_handler_.has_pending_error());
280             top->pending_error_handler_.ThrowPendingError(info->isolate(),
281                                                         info->script());
282             return false;
283         }
284     }
285     [...]
295     info->set_scope(scope);
296     return true;
297 }
298 ...

```

Listing 6.3: Scope chain traversal in *v8* source code.

By having places online to exchange information and source code for understanding the underlying functionality, was helpful when trying to verify the above mentioned explanation. In a similar way perhaps, smart devices observing physical phenomena can be enabled to use the IoT as a global infrastructure for exchanging their own observations in order to help each other make decisions. The local discovery functionality of the Home-Thing module is limited to a single subnet, but the smart devices built on it are still able to add functionality

⁶<http://stackoverflow.com/questions/2485423>

⁷<https://chromium.googlesource.com/v8/v8.git>

that allows them to communicate with the internet.

6.5.3 Summary

To propose a possible solution for distributed discovery of smart devices, a custom solution using announcement based broadcasting was implemented. The prototype developers were provided with representations of smart devices on the network through the API without experiencing problems with the connection.

The only connection issue encountered was accumulation of smart device representations. This was caused by the absence of disconnect packets from smart devices that crashed.

The non-blocking IO of Node.JS seemed to confuse some of the developers. A possible cause could be their limited experience with Node.JS. Nonetheless, the documentation can be improved by adding an explanation about this to help the smart device developers.

To add a global variable for easier iteration of peers is a feasible solution in future versions of the Home-Thing module that was proposed by one of the developers. By not using the *var* keyword when defining the variable in the local module scope, the variable becomes global unless a variable in the outer scopes use the same name.

6.6 RQ2 - Data Exchange

In the second research question I asked how web technology could be used for exchanging data and sending notifications between smart devices. I therefore had to decide which transfer protocol to use and how to structure the data. The notification capabilities were essential to allow data subscriptions to be made between smart devices.

The way data was structured in the module was inspired by Object-Oriented Programming (OOP), where each object has a number of properties containing data. Functionality for creating properties and allowing them to be shared between different smart devices was available through the module API. Properties used the same types as in JSON, in order to be compatible with the format used when using Hypertext Transfer Protocol (HTTP) to exchange data between smart devices. Furthermore, it was decided to use the Representational State Transfer (REST) architectural style. The URLs to the REST interfaces of each smart device were distributed with the service discovery packets discussed in Section 6.5.1. Notifications were sent as HTTP POST-requests, which made the subscription feature similar to Web Hooks. This allowed callbacks to be triggered by changes in properties on other smart devices, creating subscription functionality.

The developers were not exposed to the underlying functionality of the module. They were given links to the documentation for the API, and were asked to develop their own smart device prototypes during a series of workshops. Flattening of the API structure and rename the data type *text* to *string*, was improvements suggested by the smart device developers in the interviews that followed.

6.6.1 Exploration

The Home-Thing module implementation presented a number of choices between different protocols, styles and formats for transferring data. In the following sections, I will discuss my findings regarding the underlying technology from the exploratory phase of the project. This is discussed in light of the second research question.

Data Transfer

The HTTP protocol and URIs are both proven web technologies that have been used for more than two decades. Together, they are essential to the world wide web as we know it today. HTTP is a natural protocol choice for transferring data representations between machines on the web. However, when developing smart devices for constrained environments or continuous data, other more lightweight protocols such as Constrained Application Protocol (CoAP) may be more efficient.

CoAP, is expected to become a standard communication protocol for constrained smart devices [11] and is described in Section 2.5.3. Currently, there are several CoAP modules for Node.JS. Some of them were tested during the project's exploration phase. The drawback of using the *coap* module, is that it is still in beta, and the risk of finding bugs in the implementation is therefore greater. Additionally, the API may change drastically before it is out of beta, which means that the differences must be addressed as well. When creating the Home-Thing module, I concluded that it was less risk involved in relying on an implementation for a the well known protocol such as HTTP. The CoAP protocol could instead be implemented in future versions of the Home-Thing module to provide wider support for IoT devices. With this in mind, I created a structure to facilitate easy implementation of additional protocols in the future.

Service Architecture

To add restraints on how the data would be accessed through the HTTP protocol, both REST and Simple Object Access protocol (SOAP) were considered.

SOAP was originally designed for allowing software on one computer to run procedures remotely on other computers. It uses Extensible Markup Language (XML) to transfer structured data and went from being a single protocol to be a protocol framework. Prescod [43] describes that SOAP has experienced a drastic growth over the years. Due to this growth, it is now difficult to say with confidence that SOAP and REST are in competition. In the article *Roots of the REST/SOAP Debate* Prescod explains:

[...] SOAP grew and changed so drastically that now SOAP advocates can deprecate a feature (the SOAP type system) that was at one point considered the primary reason for SOAP's existence [43].

REST was built for performance, and with its simplicity and scalability it emerges as a simpler alternative to SOAP [62]. It focuses on the interaction between components and is not concerned with the syntax that is used to describe the data [22]. In his doctor of philosophy dissertation, Fielding states:

The Representational State Transfer (REST) style is an abstraction of the architectural elements within a distributed hypermedia system. REST ignores the details of component implementation and protocol syntax in order to focus on the roles of components, the constraints upon their interaction with other components, and their interpretation of significant data elements [22].

In REST, resources are manipulated through representations consisting of data and metadata [22].

Guinard et al. [26] argued that although web servers are likely to be embedded into an increasing number of devices, it can not be assumed that every smart device will be able to directly offer a RESTful interface. However, the CoAP protocol was created by the Constrained RESTful Environments (CoRE) working group for this purpose in particular. Work on CoRE aims at making the REST architecture on devices such as 8-bits microcontrollers with limited memory [50]. By using REST and HTTP for now, the module can be later upgraded to instead use the CoAP protocol to also support constrained devices.

Resource Description

In order to produce a working solution for allowing smart devices to exchange data, it was necessary to decide on a common format for the data representations. I know that both XML and JSON are common formats used in REST APIs. A possible solution could have been to implement both and let the client decide. However, in order to avoid unnecessary complications, it was decided to only implement one. JSON was therefore chosen mainly due to personal preferences and its integration in JavaScript and Node.JS. Additionally, it should be mentioned that it is considered to be more lightweight than XML [26].

Notifications

A HTTP server cannot initiate a connection. It can only send a response when it has first been requested by the client [36]. As an increasing number of interactive web pages has been introduced to the Web, a commonly used technique for pushing updates to clients has been to make them request updates with using JavaScript. A well known page where this technique is currently used is Twitter⁸, which makes the client send requests to the server every 10 seconds in order to load any new tweets. Other methods with less response time, compared to this approach, are for instance the ones known under the umbrella expression Comet. Comet use either long polling or HTTP streaming, which is based on pending HTTP requests. These requests are first responded to when the server is ready to push new content to the client [36]. The server is forced to open a number of different HTTP connections for each client, which creates unnecessary high overhead. Additionally, it can cause a significant increase in data usage [36]. More details about Comet is described in Section 2.3.3.

Smart devices based on the Home-Thing module can both be classified as clients and servers. Therefore, they could send notifications to each other using *POST* requests. A feature used by smart device developers that used notifications was property subscriptions. By registering subscriptions with the

⁸<https://twitter.com/>

property owner, smart devices were able to listen for data changes on other devices. In order to send notifications to subscribing devices, an internal list was dedicated to hold the subscription data, which described the relations between internal properties and the smart devices subscribing to them. This functioned in the same way as Web Hooks, where the subscriber sends a request containing the callback URI [30]. The Home-Thing module handled the communication details and offered a dedicated representational object for the subscriptions to the smart devices utilizing the module.

6.6.2 Feedback

In the following sections, I will discuss future improvements suggested by the workshop developers. More details about this can also be found in Section 5.5.7.

Flatten Structure

Due to the indentation depth caused by nested functions and inconsistencies, one of the developers lost control of his code during one of the workshops. The inconsistency may have been caused by a tiring number of spaces, which was required in order to maintain the right indentation level. When it reached a certain depth, some shortcuts could have been taken. Therefore, reducing the number of asynchronous calls in the API could be helpful. This could either be done by using while loops to wait for response, which could waste resources. Another solution could be to chain some of the asynchronous calls and make them act as one. One of the suggestions from the developers was to decrease the depth of the structure, by for instance removing the *getDetails()* function. Values such as device name and description could be made available through properties in the representational smart device objects. Some of the asynchronous operations could then be chained when creating the representational objects, which would make the data available through properties as suggested. The drawback, however, would be an increased amount of unnecessary network traffic in cases where the requested properties would not be used. Therefore, a static declaration of the resources used by the smart devices could be required in order to retrieve and process only the necessary data. The declaration could be similar to the *manifest-file*⁹ in Android applications, which declares restricted features with permission-tags.

Renaming of Data Type

The last suggestion listed in Section 5.5.7 was to rename the data type *text* to *String*. The developer elaborated that the target audience for the module, the smart device developers, would recognize the concept of strings. This suggestion was addressed following the third workshop with backward compatibility to avoid breakage of already existing prototypes, as the module was released on NPM prior to this change.

⁹<http://developer.android.com/guide/topics/manifest/manifest-intro.html>

6.6.3 Summary

SOAP may one day evolve to better support IoT applications. Still, it is suggested that HTTP and REST is currently a more fitting alternative. By using HTTP and REST the module demonstrated a possible way to transfer data and exchange notifications between smart devices. REST is also supported in CoAP, which may one day be the standard protocol for network related functionality in smart devices. With this in mind, the module was structured to enable easy implementation of additional protocols for communications. JSON was chosen instead of XML due to its performance and integration in JavaScript. To allow for sending of notifications between smart devices, a solution similar to Web Hooks was implemented to create subscriptions for properties. Possible improvements suggested by the smart device developers was to decrease the depth in the API structure and rename the property type *text* to *string*.

6.7 RQ3 - Streams

The third research question revolved around how smart devices can utilize streams in order to transfer continuously updated data. Smart devices should be able to receive continuous updates for resources or object representations that change over time. With the ability to send streams it becomes easier to exchange real-time sensor data without flooding the network with HTTP requests. Streams can also for instance be used to transfer audio and video and offer short response time, as it holds the connection open instead of opening a new for each chunk of data.

In order to transfer streams over the network, I used the Node.JS modules *socket.io* and *socket.io-stream* which utilize the WebSocket protocol. Smart device developers were given access to the streaming functionality through the API, in order to make the devices share streams over the network. Each stream could be piped, processed and published in order to create processing pipelines across the network.

The smart device developers from the workshops disagreed to whether or not streams were easy to use. In future versions of the module, streams can be given more focus in the documentation in order to help developers understand the concept and how it can be utilized.

6.7.1 Exploration

The Extensible Messaging and Presence Protocol (XMPP) standard, detailed in Section 2.5.2, allows for creation of XML streams in both directions over Transmission Control Protocol (TCP) connections [34, 48]. The drawback of XMPP is its complexity and that it is heavy for devices with limited resources [26]. XMPP is based on long-polling over HTTP [34], as described in Section 2.3.3.

HTTP/2.0 and WebSockets are better suited for transferring data streams than HTTP/1.1 [59]. The WebSocket Protocol, described in Section 2.5.1, was created to allow scripts in browsers to have access to bidirectional TCP connections [20]. Data sent with WebSockets can be binary or in text format. Transferring binary data is useful for instance when smart devices need to exchange video or audio. Furthermore, streams on the network can be piped, processed

and published in order to create processing pipelines across the network. This is also known as the pipe and filter style [22].

Unlike XMPP, the WebSocket protocol does not need to send HTTP headers, except for when initiating the connection with a handshake [20, 34]. Using WebSockets, therefore require less network traffic in order to send the same data when it cannot be achieved with only one HTTP request [34]. Therefore I chose WebSocket for transferring streams of continuous data. The Node.JS modules used for establishing WebSocket connections were *Socket.io*¹⁰ and *Socket.io-stream*¹¹. These modules were encapsulated by the Home-Thing module and hidden behind the API, which exposed representational objects to smart devices to allow handling of streams.

Shared port

During the workshops, the Home-Thing module used different ports for HTTP and WebSocket, due to technical issues encountered during the module development when trying to use the same port. According to the WebSocket specification [20], it should have been possible to share ports:

The opening handshake is intended to be compatible with HTTP-based server-side software and intermediaries, so that a single port can be used by both HTTP clients talking to that server and WebSockets clients talking to that server [20].

The error message that appeared when Socket.IO started, indicated that the port was already in use. When having conducted the workshops, I was able to examine the problem more closely. By connecting the first HTTP instance with Socket.IO to add the HTTP module as a parameter in the constructor of Socket.IO, the port sharing was finally accomplished.

Working with Streams

After having implemented streaming functionality in the module, I was able to successfully stream video from a camera on one computer to the video player on another, by using prototypes based on the module. I also created a Peer-to-Peer (P2P) chat by piping the input stream of the Unix Shell to the other Home-Thing module instances. This required few lines of code which is detailed in Listing 4.3. It is possible to easily create a chat client using HTTP and properties as well using the subscription feature. Still, in this case it required less effort to just use the available streams instead.

6.7.2 Feedback

Each developer managed to implement functionality based on the Home-Thing module, in order to read external streams. Still, their questions and progress during the workshops suggested that the comprehension of streams was different between the developers. A possible cause, could be that they had no prior experience with Node.JS, as detailed in Section 5.5.1. Additionally, it is also possible they are used to handle data streams in different ways. Observations

¹⁰<https://www.npmjs.com/package/socket.io>

¹¹<https://www.npmjs.com/package/socket.io-stream>

detailed in Section 5.5.6 also suggest that the developers to some extent were uncertain about how callback was used in JavaScript and when they would be executed. The learning curve was steep during the development session despite having taken part in a tutorial, detailed in Section 5.1.2, in the first part of the workshop. During an interview, one of the developers explained:

Jade: I didn't understand how to use streams. I haven't used it except in Java when handling file streams. It seemed like the same concept, but I didn't understand it completely, so that was the main problem.

Creation of empty streams proved to be a difficult task during the workshops. As mentioned in Section 5.5.4, a workaround was provided to allow creation and publishing of text streams. Without being able to find an easy solution for creating and sharing empty writable streams, the workaround used was to create a writable file stream and another stream that continuously read the file. The text written to the write stream was picked up by the read stream and published using the API in order to share it over the network. This code is detailed in Listing 6.4. After having completed the workshops, I was able to find a stream type that could have solved the problem with less code. By using *PassThrough streams*¹², the input of the stream would be directly sent to the output. This could also have been achieved with a *transform stream* with an empty transform function only passing forward the input. The used approach and the approach later discovered are detailed in Listing 6.5 and Listing 6.4.

```
1  ...
2  // Include file system module
3  var fs = require("fs");
4
5  // Create writable file stream
6  var file = fs.createWriteStream('output.log')
7
8  // Create a continuous readable file stream
9  var terminal = require('child_process');
10 var stream = terminal.spawn("tail", ["-f", "output.log"]);
11 ...
```

Listing 6.4: Streaming code, used during the workshops.

```
1  ...
2  // Create writable and readable stream
3  var stream = new require('stream').PassThrough();
4  ...
```

Listing 6.5: Alternative code for streams.

Not all developers were able to compare streams and subscriptions during the workshops. A developer having experience with both, said he thought streams were more difficult to use. Still, as detailed in the following quote, he believed streams had more potential than properties:

Ryan: My impression is that it was simpler to quickly get results using properties compared to streams. Still, I think that streams can yield better results with respect to how much time you spend on it. One can also get more interesting functionality.

¹²<https://nodejs.org/api/stream.html>

What was meant by *more interesting*, could perhaps be more novel solutions or more impressive functionality in terms of how much that could be accomplished with a only few lines of code.

Some of the questions that came up during the workshops, was how streams differed from subscriptions. As discussed in Section 6.7.1, the difference is the protocols that are used and how long the connection stays open. Subscribing to a large number of properties that change continuously, can cause the module to freeze. This was experienced by one of the developers during the first workshop, who created a smart device with hundred properties that were set to update continuously. He explained:

Ryan: I'm not sure what to do now. The monitor has crashed by the way. I think it has something to do with that 1 millisecond interval thing.

Without the HTTP overhead created by the updates from property subscriptions, streams could have been utilized to improve the performance. Perhaps, in future versions of the Home-Thing module, a feature could be implemented to automatically switch from using HTTP to streams streams when updates are frequent and above a certain threshold value found in performance tests. Nonetheless, the choice between subscriptions and streams, when using the module today, may affect the performance of the smart devices in regard to power consumption and network usage. Adding more details in the module documentation about this may improve it and create more depth as requested by the developers.

From another perspective, maybe it is the structure of properties and streams that should be modified. Perhaps, a better structure could be to only have properties and allow internal streams to instead be piped the representational property objects. The underlying functionality of the Home-Thing module could then automatically decide whether it would transfer the data with HTTP or WebSocket as mentioned earlier.

6.7.3 Summary

Streams can make it easier to send real-time data in networks of Home-Thing module based smart devices. In addition to having a Web interface for HTTP, which was discussed in Section 6.6, the module also hosted WebSocket. During the evaluation, HTTP and WebSocket used distinct ports, which was later changed to the same. The developers from the three workshops disagreed to whether or not the streaming functionality in the module was easy to use. The documentation has room for improvements and can help developers learn more about how streams work. Optionally, the streams could be phased out, leaving properties to handle internal streams. How to transfer the continuous data or data states could then be decided by the underlying functionality of the prototype to optimize the performance and cause less confusion for developers.

Each of the developers from the workshops, were able to read external streams from other smart devices in the course of the workshops.

In this section, the results from the exploration phase and the prototype evaluation have been discussed. In the following section, I will highlight the most important findings and conclude.

Chapter 7

Conclusion

Choice. The problem is choice.

- Neo

In this project, I have combined exploration and qualitative evaluation to create a prototype, for abstracting communication related functionality of smart devices. Abstracting communication related functionality from the rest of the smart device software, allows for a separation of concerns. Experts in network communication can for instance handle security related features and other developers can concentrate on the remaining parts of the smart device development.

7.1 Contribution

In this project, I have created a module for Node.JS which was evaluated with developers in a series of workshops. By encapsulating functionality and exposing an API, the module provides a possible solution for how to separate smart device software from its underlying communication functionality. The API allows smart devices to exchange data and communicate through local JavaScript objects, which represents other smart devices and their respective data resources. Additionally, it offers events for device discovery and subscriptions allowing callbacks to be triggered when new smart devices are discovered and their data is changed. The module was positively received by the developers and described as being simple to use.

7.1.1 Research Question 1 - Service Discovery

Utilizing UDP broadcasting, a custom implementation for announcement based service discovery, is used in the module in order to find similar smart devices on the network. Each module instance listens to broadcast packets and responds by UDP unicast. This enables service discovery with built-in URL exchange, allowing smart devices to find each other and share information about available resources. The implementation proved to be stable and predictable in local wired networks. Each of the developers managed to utilize the listeners for service discovery, triggered when new smart devices were discovered, in order to filter out specific resources.

7.1.2 Research Question 2 - Properties and Notifications

In order to share of resources, the module used used HTTP and REST. With inspiration from OOP, each smart device defines its own properties that holds data. The developers appreciated this approach. During the evaluation, properties were described as being simpler to get quick result with compared to streams. By sending HTTP POST requests, the module enabled notifications to dedicated paths, defined by each receiving smart device in advance. This allowed for creation of a property subscription feature, exposed to the developers through the API, which triggered callbacks for specified update events.

7.1.3 Research Question 3 - Streams

To also facilitate sharing of continuously updated data, the WebSocket protocol was chosen. This allowed binary and text streams to be shared between smart devices on the same port as HTTP. The developers disagreed to whether streams were easy to use. Questions from the workshops suggests that the line between streams and subscriptions should be made more clear. A possible enhancement could be to phase out the distinction between shared streams and shared properties in order to allow the module to decide whether it use HTTP or WebSocket in order to share the data.

7.2 Reflection

The evaluation involved developers in three workshops. The goal was to let them build smart devices based on the module and give feedback in semi-structured interviews that followed. To analyze the developers thoughts and questions, video and audio was recorded and later transcribed. In addition, I also offered the developers help when they had questions about the module. Being an observer, offering expert help and administrating devices on the network at the same time, was a challenge. However, the video and audio recordings helped to later discover details, that I originally missed during the workshop. In future evaluations, screen recording can also be helpful in order to watch the progress of the development more closely.

7.2.1 Selection of Methods

In this project, I used exploration and qualitative evaluation to first familiarize myself with the IoT through research literature and creation a prototype. Further, I involved developers in a series of workshops where the prototype was evaluated. The freedom of exploration allowed me to discover challenges that I found interesting and work on a possible solution for overcoming them. Evaluation with other developers allowed me to get feedback from other points of view and yielded valuable suggestions for future improvements of the module. The combination of methods was successful. It allowed me to encounter a wide spectrum of research before refining the project's focus to a precise set of challenges.

7.3 Future Work

The module has shown potential as a building block for Web based smart devices. It was received positively by the developers involved in the evaluation process. Still, it has room for improvements. With the separation of underlying transfer protocols and the other smart device functionality, the module can be improved in the future to be more secure and efficient without breaking functionality of dependent smart devices in the process.

The documentation, can be given more example code and more information about the underlying functionality of the module. Code that is not directly a part of the module, such as creation of streams can also be added in order to further support developers developers.

Using WebSocket to transfer frequent subscription updates can decrease overhead and improve the performance. Additionally, by enabling the module to automatically switch between HTTP and WebSockets in the background, less effort will be needed from the developers to decide between subscriptions and streams. This could be improved further by phasing out the current distinction between streams and properties.

Bibliography

- [1] Ian Alexander and Neil Maiden. Scenarios, stories, and use cases: the modern basis for system development. *Computing and Control Engineering*, 15(5):24–29, 2004.
- [2] Luigi Atzori, Antonio Iera, and Giacomo Morabito. The internet of things: A survey. *Computer networks*, 54(15):2787–2805, 2010.
- [3] Ceasar Bautista, Aditya Dahiya, Kyle Hardgrave, and David Xu. Javascript documentation generation through semantic code analysis.
- [4] Edvard Befring. *Forskningsmetoder i utdanningsvitenskap*. Cappelen Damm akademisk, 2015.
- [5] Lucia Benaquisto and LM Given. *The sage encyclopedia of qualitative research methods*, 2008.
- [6] Bruce Lawrence Berg, Howard Lune, and Howard Lune. *Qualitative research methods for the social sciences*, volume 5. Pearson Boston, MA, 2004.
- [7] Ljerka Beus-Dukic and Ian Alexander. Learning how to discover requirements. In *Requirements Engineering Education and Training, 2008. REET'08.*, pages 12–14. IEEE, 2008.
- [8] Michael Blackstock and Rodger Lea. Wotkit: A lightweight toolkit for the web of things. In *Proceedings of the Third International Workshop on the Web of Things, WOT '12*, pages 3:1–3:6, New York, NY, USA, 2012. ACM.
- [9] G er ome Bovet and Jean Hennebert. Offering web-of-things connectivity to building networks. In *Proceedings of the 2013 ACM Conference on Pervasive and Ubiquitous Computing Adjunct Publication, UbiComp '13 Adjunct*, pages 1555–1564, New York, NY, USA, 2013. ACM.
- [10] S. Cheshire and M. Krochmal. Multicast dns. RFC 6762, RFC Editor, February 2013. <http://www.rfc-editor.org/rfc/rfc6762.txt>.
- [11] Simone Cirani, Luca Davoli, Gianluigi Ferrari, R emy L eone, Paolo Medagliani, Marco Picone, and Luca Veltri. A scalable and self-configuring architecture for service discovery in the internet of things. 2014.
- [12] Walter Colitti, Kris Steenhaut, and Niccol  De Caro. Integrating wireless sensor networks with the web. *Extending the Internet to Low power and Lossy Networks (IP+ SN 2011)*, 2011.

- [13] Juliet Corbin. *Basics of qualitative research : techniques and procedures for developing grounded theory*. Sage Publications, Los Angeles, Calif, 2008.
- [14] Liviu-Gabriel Cretu. Smart cities design using event-driven paradigm and semantic web. *Informatica Economica*, 16(4):57–67, 2012.
- [15] Steve Deering. Host extensions for ip multicasting. STD 5, RFC Editor, August 1989. <http://www.rfc-editor.org/rfc/rfc1112.txt>.
- [16] David Douglas. Inductive theory generation: A grounded approach to business inquiry. *Electronic Journal of Business Research Methods*, 2(1):47–54, 2003.
- [17] Paul Dourish. What we talk about when we talk about context. *Personal Ubiquitous Comput.*, 8(1):19–30, February 2004.
- [18] W Keith Edwards, Rebecca E Grinter, Ratul Mahajan, and David Wetherall. Advancing the state of home networking. *Communications of the ACM*, 54(6):62–71, 2011.
- [19] Satu Elo and Helvi Kyngäs. The qualitative content analysis process. *Journal of advanced nursing*, 62(1):107–115, 2008.
- [20] I. Fette and A. Melnikov. The websocket protocol. RFC 6455, RFC Editor, December 2011. <http://www.rfc-editor.org/rfc/rfc6455.txt>.
- [21] Roy T. Fielding, James Gettys, Jeffrey C. Mogul, Henrik Frystyk Nielsen, Larry Masinter, Paul J. Leach, and Tim Berners-Lee. Hypertext transfer protocol – http/1.1. RFC 2616, RFC Editor, June 1999. <http://www.rfc-editor.org/rfc/rfc2616.txt>.
- [22] Roy Thomas Fielding. *Architectural styles and the design of network-based software architectures*. PhD thesis, University of California, Irvine, 2000.
- [23] David Flanagan. *JavaScript: the definitive guide*. ” O’Reilly Media, Inc.”, 2002.
- [24] Uwe Flick. *The SAGE handbook of qualitative data analysis*. SAGE, Los Angeles, 2014.
- [25] Kentaro Go and John M Carroll. The blind men and the elephant: Views of scenario-based system design. *interactions*, 11(6):44–53, 2004.
- [26] Dominique Guinard, Vlad Trifa, Friedemann Mattern, and Erik Wilde. From the internet of things to the web of things: Resource-oriented architecture and best practices. In *Architecting the Internet of Things*, pages 97–129. Springer, 2011.
- [27] Erik Guttman. Autoconfiguration for ip networking: Enabling local communication. *Internet Computing, IEEE*, 5(3):81–86, 2001.
- [28] Stephan Haller, Stamatis Karnouskos, and Christoph Schroth. *The internet of things in an enterprise context*. Springer, 2009.
- [29] PH Jarvinen. Research questions guiding selection of an appropriate research method. *ECIS 2000 Proceedings*, page 26, 2000.

- [30] Andreas Kamilaris, Andreas Pitsillides, and Vlad Trifa. The smart home meets the web of things. *International Journal of Ad Hoc and Ubiquitous Computing*, 7(3):145–154, 2011.
- [31] Shahedul Huq Khandkar. Open coding. *University of Calgary, October*, 23, 2009.
- [32] Gerd Kortuem, Fahim Kawsar, Daniel Fitton, and Vasughi Sundramoorthy. Smart objects as building blocks for the internet of things. *Internet Computing, IEEE*, 14(1):44–51, 2010.
- [33] Matthias Kovatsch, Simon Duquennoy, and Adam Dunkels. A low-power coap for contiki. In *Mobile Adhoc and Sensor Systems (MASS), 2011 IEEE 8th International Conference on*, pages 855–860. IEEE, 2011.
- [34] Markku Laine and Kalle Säilä. Performance evaluation of xmpp on the web. Technical report, Aalto University Technical Report, 2012.
- [35] Paul J. Leach, Michael Mealling, and Rich Salz. A universally unique identifier (uuid) urn namespace. RFC 4122, RFC Editor, July 2005. <http://www.rfc-editor.org/rfc/rfc4122.txt>.
- [36] S. Loreto, P. Saint-Andre, S. Salsano, and G. Wilkins. Known issues and best practices for the use of long polling and streaming in bidirectional http. RFC 6202, RFC Editor, April 2011. <http://www.rfc-editor.org/rfc/rfc6202.txt>.
- [37] Chia-Wen Lu, Shu-Cheng Li, and Quincy Wu. Interconnecting zigbee and 6lowpan wireless sensor networks for smart grid applications. In *Sensing Technology (ICST), 2011 Fifth International Conference on*, pages 267–272. IEEE, 2011.
- [38] Salvatore T March and Gerald F Smith. Design and natural science research on information technology. *Decision support systems*, 15(4):251–266, 1995.
- [39] Daniele Miorandi, Sabrina Sicari, Francesco De Pellegrini, and Imrich Chlamtac. Internet of things: Vision, applications and research challenges. *Ad Hoc Networks*, 10(7):1497–1516, 2012.
- [40] M. Nottingham and E. Hammer-Lahav. Defining well-known uniform resource identifiers (uris). RFC 5785, RFC Editor, April 2010. <http://www.rfc-editor.org/rfc/rfc5785.txt>.
- [41] Jay F Nunamaker Jr and Minder Chen. Systems development in information systems research. In *System Sciences, 1990., Proceedings of the Twenty-Third Annual Hawaii International Conference on*, volume 3, pages 631–640. IEEE, 1990.
- [42] Erika Shehan Poole, Marshini Chetty, Rebecca E Grinter, and W Keith Edwards. More than meets the eye: transforming the user experience of home network management. In *Proceedings of the 7th ACM conference on Designing interactive systems*, pages 455–464. ACM, 2008.
- [43] Paul Prescod. Roots of the rest/soap debate. In *Extreme Markup Languages®*. Citeseer, 2002.

- [44] Guillermo Rauch. *Smashing Node.js: JavaScript Everywhere*. John Wiley & Sons, 2012.
- [45] Mary Beth Rosson and John M Carroll. Scenario based design. *Human-computer interaction*. Boca Raton, FL, pages 145–162, 2009.
- [46] Andreas Ruppen, Jacques Pasquier, and T HÁ? rlimann. A restful architecture for integrating decomposable delayed services within the web of things. *International Journal of Internet Protocol Technology*, 6(4):247–259, 2011.
- [47] Leila Ben Saad, Cedric Chauvenet, Bernard Tourancheau, et al. Heterogeneous ipv6 infrastructure for smart energy efficient building. In *SDEWES*, 2011.
- [48] P. Saint-Andre. Extensible messaging and presence protocol (xmpp): Core. RFC 6120, RFC Editor, March 2011. <http://www.rfc-editor.org/rfc/rfc6120.txt>.
- [49] P. Saint-Andre. Extensible messaging and presence protocol (xmpp): Instant messaging and presence. RFC 6121, RFC Editor, March 2011. <http://www.rfc-editor.org/rfc/rfc6121.txt>.
- [50] Z. Shelby, K. Hartke, and C. Bormann. The constrained application protocol (coap). RFC 7252, RFC Editor, June 2014. <http://www.rfc-editor.org/rfc/rfc7252.txt>.
- [51] Z Shelby, K Hartke, C Bormann, and B Frank. Constrained application protocol (coap), draft-ietf-core-coap-13. *Orlando: The Internet Engineering Task Force-IETF, Dec*, 2012.
- [52] Zach Shelby, Klaus Hartke, and Carsten Bormann. Constrained application protocol (coap). Internet-Draft draft-ietf-core-coap-18, IETF Secretariat, June 2013. <http://www.ietf.org/internet-drafts/draft-ietf-core-coap-18.txt>.
- [53] Patricia M Shields and Hassan Tajalli. Intermediate theory: The missing link in successful student scholarship. *Journal of Public Affairs Education*, pages 313–334, 2006.
- [54] P.M. Shields and N. Rangarajan. *A Playbook for Research Methods: Integrating Conceptual Frameworks and Project Management*. New Forums scholarly writing series. New Forum Press, 2013.
- [55] Aleksandar Siljanovski, Anuj Sehgal, and Jürgen Schönwälder. Service discovery in resource constrained networks using multicast dns.
- [56] Pedro Teixeira. *Instant Node.js Starter*. Packt Publishing Ltd, 2013.
- [57] Viswanath Venkatesh and Susan A Brown. A longitudinal investigation of personal computers in homes: adoption determinants and emerging challenges. *MIS quarterly*, pages 71–102, 2001.
- [58] Tom Wengraf. *Qualitative research interviewing: Biographic narrative and semi-structured methods*. Sage, 2001.

- [59] Erik Wilde, Florian Michahelles, and Stefan Lüder. Leveraging the web platform for the web of things: Position paper for w3c workshop on the web of things. 2014.
- [60] Jia Zhao, Chunming Qiao, Raghuram S Sudhaakar, and Seokhoon Yoon. Improve efficiency and reliability in single-hop wsns with transmit-only nodes. *Parallel and Distributed Systems, IEEE Transactions on*, 24(3):520–534, 2013.
- [61] Fen Zhu, Matt W Mutka, and Lionel M Ni. Service discovery in pervasive computing environments. *IEEE Pervasive computing*, 4(4):81–90, 2005.
- [62] Joe Zou, Jing Mei, and Yan Wang. From representational state transfer to accountable state transfer architecture. In *Web Services (ICWS), 2010 IEEE International Conference on*, pages 299–306. IEEE, 2010.

Glossary

Arch Linux ARM A Linux based Operating system¹ which was used on Raspberry Pis in this project during the development workshops.

C++ Object oriented programming language created by Bjarne Stroustrup.

Comet Umbrella term for HTTP based server-client data pushing.

Controlling the Home Earlier relevant project with focus on mobile interfaces. This is further described in Section 1.1.1.

GitHub Web based Git repository² where most modules for Node.JS is hosted [44].

Home-Thing The prototype module³ created in this project, based on Node.JS. The module is described in Chapter 4. Results from testing is detailed in Chapter 5.

JavaScript Programming language often used in browsers, but also in Node.JS.

node-uuid Module for Node.JS that generate UUIDs of version 1 or version 4 based on RFC4122 [35].

Node.JS Platform built on Google's v8 JavaScript engine.

NPM Package manager for Node.JS that can download, install and publish modules.

Raspberry PI Low cost credit card sized computer⁴ which is able to run Arch Linux ARM for instance.

Socket.IO Node.JS module which use the WebSocket protocol to host a bidirectional socket connection.

Unix Shell Command-line interpreter for Unix and Unix like Operating systems (OSs).

¹<http://www.archlinuxarm.org/>

²<https://www.github.com/>

³<https://github.com/anskaal/home-thing>

⁴<https://www.raspberrypi.org/>

v8 Open source JavaScript engine used in Google Chrome and Node.JS [44, 56].

Web Hook Subscription feature allowing servers to subscribe to events on other servers [30].

WebSocket Protocol for transferring streams specified in [20]. This protocol is further described in Section 2.5.1.

Acronyms

6LoWPAN IPv6 over Low power Wireless Personal Area Networks.

ALSA Advanced Linux Sound Architecture.

API Application Programming Interface.

CoAP Constrained Application Protocol.

CoRE Constrained RESTful Environments.

DNS Domain Name System.

GUID Globally unique identifier.

HCI Human–computer interaction.

HTTP Hypertext Transfer Protocol.

IDE Integrated Development Environment.

IETF Internet Engineering Task Force.

IoT Internet of Things.

IP Internet Protocol.

IT Information Technology.

JSON JavaScript Object Notation.

M2M Machine to Machine.

mDNS Multicast DNS.

OOP Object-Oriented Programming.

OS Operating system.

P2P Peer-to-Peer.

PD Participatory Design.

REST Representational State Transfer.

RFID Radio-Frequency Identification.

SOAP Simple Object Access protocol.

SSH Secure Shell.

TCP Transmission Control Protocol.

UDP User Datagram Protocol.

URI Uniform Resource Identifier.

URL Uniform Resource Locator.

UUID Universally unique identifier.

WoT Web of Things.

WSN Wireless Sensor Network.

XML Extensible Markup Language.

XMPP Extensible Messaging and Presence Protocol.

Appendix A

Documentation

home-thing

A module for npm

Install

From git:

```
npm install
```

How to set up

```
var name = "Example Thing";
var description = "A smart device that is being tested.";

var app = require("home-thing")(name, description);
```

How to get the value of a property called value on another device:

```
var propertyName = "volume";

app.on("new peer", function(peer){

  peer.getProperties(function(property){

    property[propertyName].get(function(value){

      console.log(propertyName + " = " + value);

    });

  });

});
```

Structure

- HomeThing
- Peer
- Property
- Stream
- Subscription

HomeThing

```
var name = "My Smart Device";
var description = "This is the coolest Smart Device";

var app = require("home-thing")(name, description);
```

This is the main-object that should only exist in one instance on the computer. The technical reason for this, is that the discovery mechanism requires reservation of port 50000. In the examples of the following documentation, the variable name `app` is used for holding this object.

Methods

- `addProperty()`
- `addStream()`
- `getPeers()`
- `end()`
- `on()`
- `removeProperty()`
- `removeStream()`

Add Property

```
object addProperty (name, type, description, methods)
```

Type	Parameter	Description
String	<code>name</code>	Name of the property
String	<code>type</code>	"boolean", "number", "text", "list", "object"
String	<code>description</code>	A short textual description of the property.
Object	<code>methods</code>	Object with "get" and "set" functions.

Without anything to share, smart devices are not very interesting. The `addProperty` method allows adding of properties like temperature and power state for instance.

In order to share a property with the other smart devices on the network, a `get`-function must be declared. This function must be able to return the internal property value when requested by other peers. If you want to enable other devices to also change the value, you must also provide a `set`-function. For more information about this, see the examples below.

In addition to the functions, the property's name, type and description must also be provided according to the table above.

Note that property names are not case sensitive. Also, the object returned contains `get`- and `set`-functions similarly to what you declared except for one important difference; they also notify subscribing peers' listeners when the value change.

Example 1

```
// This is the variable that will be made externally available.
```

```

var volume = 0;

var volumeProperty = app.addProperty("volume", "number", "Volume of computer", {

  // The get method must use a callback method when returning the value.
  'get' : function(callback){
    callback(volume);
  },

  // The set method must also use a callback method.
  'set' : function(value, callback){
    volume = value;
    callback();
  }
});

```

Another way to do the same:

Example 2

```

function VolumeProperty(){

  // This is the variable that will be made externally available.
  var internalValue = 0;

  // The get method must use a callback method when returning the value.
  this.get = function(callback){
    callback(internalValue);
  };

  // The set method must also use a callback method.
  this.set = function(volume, callback){
    internalValue = volume;
    callback();
  };

}

var name = "volume";
var type = "number";
var desc = "Volume of computer";

var volumeProperty = app.addProperty(name, type, desc, new VolumeProperty());

```

Add Stream

```
void addStream (name, description, stream)
```

Type	Parameter	Description
String	name	Name of the stream.
String	description	A short textual description of the stream.
Object	stream	The stream-object containing stdout.

Not all data can be described sufficiently by properties. Therefore, this module are able to share streams over the WebSocket protocol. The addStream method allows your device to share internal data streams

with other devices. This can for instance be a video streamed from a web camera or the output of a bash script. Sharing of streams enables creation of processing pipelines across the network.

Example

```
// This allows us to spawn child processes.
var terminal = require('child_process');

// We start ffmpeg which will access the video device and pipe it to stdout in avi format
var stream = terminal.spawn("ffmpeg", ["-i", "/dev/video0", "-f", "avi", "-"]);

// Publishing the stream, making it visible for other devices on the network
app.addStream("video", "Video from webcam", stream.stdout);
```



End

```
void end ()
```

The end method shuts down the module and sends disconnect signal to all devices on the network.

Get Peers

```
Array getPeers ()
```

The getPeers method returns all available peers on the network as objects in an array from an internal cache. The peers returned here are smart devices that responded to the network discovery on startup in addition to smart devices that later broadcasted their presence on the network.

Example

```
var peers = app.getPeers();

// Print out Array of peer objects
console.dir(peers);
```

On

```
void on (eventName, callback(object))
```

Type	Parameter	Description
String	eventName	"new peer"
Function	callback	A function that will run when event is triggered.

The on method allows you to build event driven execution of tasks. The supported listeners are listed in the table above.

Example

```
app.on("new peer", function(peer){  
    // Prints peer objects of all new peers  
    console.dir(peer);  
});
```

Remove Property

```
void removeProperty (name)
```

Type	Parameter	Description
String	name	The name of the property

The removeProperty method allows you to remove previously created properties.

Example

```
// Removing a previously added property called volume  
app.removeProperty("volume");
```

Remove Stream

```
void removeStream (name)
```

Type	Parameter	Description
String	name	The name of the stream

The removeStream method allows you to remove a published stream.

Example

```
// Removing a previously added stream called video  
app.removeStream("video");
```

Peer

The Peer object represents a smart device on the network.

```
var peers = app.getPeers();  
for(var peer in peers){  
    ...  
}
```

Methods

- `addSubscription()`
- `getDetails()`
- `getProperties()`
- `getStreams()`
- `on()`

Add Subscription

`Object` `addSubscription` (property, callback)

Type	Parameter	Description
String	property	Name of the property
Function	callback	The function that will run when property is changed.

The `addSubscription` method enables a smart device to subscribe to changes of a property on another device.

Example

```
peer.addSubscription("volume", function(data){
  // Prints out new data every time a update notification is received.
  console.dir(data);
});
```

Get Details

`void` `getDetails` (callback(object))

Type	Parameter	Description
Function	callback	The function that will run when the new value returns.

The `getDetails` method requests basic details such as name and description from the smart device.

Example

```
peer.getDetails(function(details){
  // Prints out peer's basic details like name and description.
  console.dir(details);
});
```

Get Properties

`void` `getProperties` (callback(Array))

Type	Parameter	Description
Function	callback	The function that will run when the details returns.

The `getProperties` method returns an array of `Property` objects.

Example

```
peer.getProperties(function(properties){
  // Prints out peer's properties.
  console.dir(properties);
});
```

Get Streams

```
void getStreams (callback(object))
```

Type	Parameter	Description
Function	callback	The function that will run when a list of streams returns.

Example

```
peer.getStreams(function(streams){
  // Prints out peer's published streams.
  console.dir(streams);
});
```

On

```
void on (eventName, callback)
```

Type	Parameter	Description
String	eventName	"disconnect", "structure change"
Function	callback	The function that will run when the event is triggered.

Example

```
peer.on("disconnected", function(){
  console.log("This peer just disconnected");
});
```

Property

The `property` object is a local handler for an external resource.

```
peer.getProperties(function(properties){
  if(properties && properties["volume"]){
    properties["volume"].get(function(value){
      console.log("Volume is "+ value);
    });
  }
});
```

Methods

- get()
- set()

Get

```
void get (callback(value))
```

Type	Parameter	Description
Function	callback	The function that will get the property.

This method lets you request a property value from its belonging smart device.

```
volumeProperty.get(function(volume){
  // Prints the current value of the volume property
  console.log("Volume is now "+ volume);
});
```

Set

```
void set (value, callback(object))
```

Type	Parameter	Description
Var	value	The value to set
Function	callback	The function that will change the property.

The set method sends the given value to the smart device that has the property. The callback function is executed when the smart device has given its response.

```
volumeProperty.set(54, function(){
  console.log("Volume set.");
});
```

Stream

```
peer.getStreams(function(streams){
  for(var key in streams){
    console.log("stream "+ key);
  }
});
```

Methods

- on()
- pipe()
- unpipe()

On

`void on (eventName, callback)`

Type	Parameter	Description
String	eventName	"data", "end" (see nodejs-docs)
Function	callback	The function that will run when the event is triggered.

Example

```
stream.on("data", function(data){
  console.log("line: "+ data);
});
```

Pipe

`void pipe(outputStream)`

Type	Parameter	Description
Object	outputStream	The stream that will receive data.

The pipe method lets you pipe streams from other peers to your local processes.

```
// Saving stream from other device to a file
var fileStream = fs.createWriteStream(filename);
remoteVideoStream.pipe(fileStream);
```

Unpipe

`void unpipe()`

The unpipe method breaks the pipe between two streams.

```
stream.unpipe();
```

Subscription

```
var subscription = peer.addSubscription("volume", function(data){
  console.dir(data);
});

subscription.on("added", function(){
  console.dir("Added Subscription!");
});
```

Subscriptions

Methods

- end()
- on()

End

`void end (callback)`

Type	Parameter	Description
Function	callback	The function that will run when the subscription is ended.

End the subscription.

```
subscription.end(function(){
  console.log("Subscription ended.");
});
```

On

`void on (eventName, callback(object))`

Type	Parameter	Description
String	eventName	"added"
Function	callback	The function that will run.

```
var subscription = peer.addSubscription("volume", function(data){
  console.log("Subscription added.");
  console.dir(data);
});
```